REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* June 2012	**2. REPORT TYPE** Other (User's Manual)	**3. DATES COVERED** *(From - To)* June 2012 – August 2012

4. TITLE AND SUBTITLE The CRUNCH Suite of Atomic and Molecular Structure Programs	**5a. CONTRACT NUMBER** In-House
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Jeffrey D. Mills and Gordon A. Gallup	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 50260541
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory (AFMC) AFRL/RQRP 10 E. Saturn Blvd. Edwards AFB CA 93524-7680	**8. PERFORMING ORGANIZATION REPORT NO.**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory (AFMC) AFRL/RQR 5 Pollux Drive Edwards AFB CA 93524-7048	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **AFRL-RQ-ED-OT-2012-262**

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution A: Approved for Public Release; Distribution Unlimited. PA#12805

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This experimental study focused on the coupling of transverse acoustic flow perturbations with two different fundamental phenomena that take place in combustion chambers: reactive, condensed-phase droplet combustion, and non-reactive shear-coaxial injection flows. The study on fuel droplet combustion characteristics examined the response and behavior of various burning fuel droplets during exposure to external acoustical perturbations. These liquid fuels included ethanol, methanol, aviation fuel (JP-8), liquid synthetic fuel derived from natural gas, and a blend of JP-8 and synfuel. The study examined combustion during excitation conditions in a closed waveguide in which the droplet was situated at or near a pressure node, where the droplet experienced the greatest effects of velocity perturbations, and at or near a velocity node (pressure antinode), where the droplet was exposed to minimal velocity fluctuations but maximum pressure fluctuations. A two-speaker configuration provided the means to produce a fairly symmetric acoustic field in the waveguide. In the absence of acoustic excitation, values of the measured droplet burning rate constant *K* were generally consistent with available values for the different fuels explored. During acoustic excitation of droplets situated in the vicinity of ! a pressure node or antinode, flame orientation was consistent with the ! sign of an acoustic radiation force acting on the burning system, creating conditions ! where the flame deflection switched, depending on the relative location of the droplet. !
The acceleration associated with the acoustic radiation force was estimated by measuring the degree of deflection that the flame underwent relative to an unforced flame. Although overall there were no significant variations in the measured *K* values with changing acoustic excitation, in some cases, locally increased *K* values were observed to be associated with larger measured acoustic accelerations. This study also examined the extinction characteristics and made preliminary estimations of the extinction strain rates of the different fuels.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Wayne Kalliomaa
a. REPORT **Unclassified**	**b. ABSTRACT** **Unclassified**	**c. THIS PAGE** **Unclassified**	SAR	44	**19b. TELEPHONE NO** *(include area code)* 661-525-6442

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

The CRUNCH suite of atomic and molecular structure programs.

User's Manual

Jeffrey. D. Mills
Air Force Research Laboratory
10 East Saturn Blvd.
Edwards Air Force Base, CA 93524-7680
`Jeffrey.Mills@edwards.af.mil`
`jeffmills@mailaps.org`

and

Gordon. A. Gallup
Department of Physics and Astronomy
University of Nebraska-Lincoln
Lincoln NE 68588-0299
`ggallup@unlserve.unl.edu`
`http://www.unl.edu/ggallup`

August 13, 2012

Contents

1 Introduction

The CRUNCH suite of atomic and molecular structure calculation programs has been worked on by students and faculty at the University of Nebraska over several decades. Using FORTRAN, the project was started in the late 1960s. Over the years sections were added in that language until 1988-1992 when a transition was made to C, for greater ease of use in a UNIX® environment. Development has continued until today, with usage currently limited to UNIX® platforms.

A comment concerning the name seems appropriate. As the program segments were originally being written in the late 1960s, one of the computing center operators dubbed them "Cap'n Crunch®" morsels (the breakfast cereal was new at the time) because of their hogging of computer resources. The name stuck, but without the "Cap'n".

These programs can be used to calculate, for a particular geometry, wave functions and energies at a number of levels of theory. These include restricted Hartree-Fock (RHF) for closed-shell species, restricted open-shell Hartree-Fock (ROHF), unrestricted Hartree-Fock (UHF), multiconfigurational self-consistent field (MCSCF), and configuration interaction (CI) using either orthogonal or nonorthogonal orbitals. (These two types of CI are often called multireference CI (MRCI) and multiconfigurational valence bond (MCVB)[1], respectively, and are sometimes referred to as orthogonal CI and nonorthogonal CI in the remainder of this manual.) Several smaller sections provide properties such as orbital and wave-function values and electron density, second-order Møller-Plesset (MP2) energies, electrostatic moments, and, in addition, dipole transition moments. Configuration population may be analyzed and natural orbitals may be computed from the CI calculations. The MCVB outputs can, when desired, be converted to a Heitler-London-Slater-Pauling (HLSP) valence-bond basis. A simple, but robust, optimizer is also included so that geometry (and other) optimizations can be performed. Many of the more common methods are described in reference [2].

Highlights of Unique or Noteworthy Features

- The CI modules can handle either orthogonal or nonorthogonal orbitals in the configurations.
- The sophisticated configuration selector makes explicit use of the symmetric group to directly enforce electron antisymmetry on individual configurations and will also carry out spatial symmetry projection so that the Hamiltonian matrix may contain one or more symmetry species.
- Hartree-Fock states for the lowest energy of each symmetry species (not just the ground state) may be determined. State averaging is also available at the Hartree-Fock level.
- Heitler-London-Slater Pauling valence-bond states can be obtained from the MCVB results.
- Calculations of systems with cylindrical symmetry may be performed in the continuous linear point groups, $\mathrm{C}_{\infty\mathrm{v}}$ and $\mathrm{D}_{\infty\mathrm{h}}$ at many levels. Hartree-Fock treatments of atoms in full spherical symmetry are supported.

In contrast to many monolithic quantum chemical suites, CRUNCH is organized as a number of independent modules, each of which is an executable that can be run by itself. This provides the user with maximum flexibility in composing a particular calculation. However, in normal practice one uses the C-shell script, `crunch`, that is provided to make a computational sequence more automatic and easily repeated. Nevertheless, even script users must have sufficient knowledge to be able to use the modules in the correct order. The example problems and the recent monograph[3] are especially recommended.

The modules in CRUNCH may be sorted or classified in a number of ways. For the new user it is perhaps most helpful to initially focus on those which provide core functionality, deferring consideration of secondary modules which calculate properties and others which perform utility or support duties. This three-fold division pervades the remainder of this manual. However, before taking up the details of input preparation for the most central modules, the next section continues with a brief description of the basic functionality of all of the modules. There follows, then, some general comments concerning usage and input files and an important discussion of the use of symmetry throughout CRUNCH.

2 Functionality by Module, Briefly Described

Primary Modules:

The principal CRUNCH modules can be divided into those involving integral generation, Hartree-Fock (HF) calculation, orbital transformation, configuration selection and symmetry projection, orthogonal and nonorthogonal CI, matrix eigensolution, and parameter optimization. We list here each module with a more-or-less mnemonic title and a short description.

- Integral modules

  `lobeWat` Lobe Gaussian integrals. Produces one- and two-electron integrals for atoms and molecules using lobe Gaussians.

  `atmintc` Atomic Slater integrals. Produces one- and two-electron integrals for single atoms using Slater orbitals.

- Hartree-Fock modules

  `gscfnn` SCF calculation. Uses the products of either of the integral programs to carry out a spin-restricted, closed- or open-shell Hartree-Fock (RHF or ROHF, combined in the acronym R(O)HF for brevity) calculation. It can determine the lowest energy and wave function of each symmetry type. State-averaged wave functions may also be obtained. Multipole electrostatic moments are also determined.

  `mxsscf` Maximum-spin R(O)HF calculation. This alternative allows greater user control over convergence for systems in which the total spin is the largest allowed by the number of singly occupied orbitals (including closed-shell systems).

  `uhfnn` UHF calculation. This performs a spin-unrestricted Hartree-Fock calculation.

- Orbital-transformation modules

  `matmld` Orbital matrix melding. Combines multiple orbital basis sets to prepare for the transformation of integrals in terms of atomic orbitals (AOs) to integrals over molecular orbitals (MOs).

  `trannn` Integral transformation. Transforms one- and two-electron integrals over atomic orbitals to integrals over molecular orbitals (or even more general, user-specified orbitals).

  `ctran` Transformation to complex orbitals. A special-purpose integral transformation program for systems of cylindrical symmetry ($C_{\infty v}$ or $D_{\infty h}$).

- Configuration and symmetry module

  `symgenn` Symmetry generation and configuration selection. Generates a set of symmetry-restricted configurations for use in a CI calculation. Supports both discrete and continuous ($C_{\infty v}$ or $D_{\infty h}$) point groups.

- Orthogonal-orbital configuration-interaction modules

  `mp2` MP2 calculation. Carries out a Møller-Plesset, second-order perturbation correction of R(O)HF wave functions.

  `omtrcIII` MRCI matrix generation. This is the current orthogonal-orbital CI matrix generator. It supports discrete and continuous point group symmetries.

  `mcscf` MCSCF calculation. Performs a multiconfigurational, self-consistent-field calculation.

- Nonorthogonal CI matrix module

  `nomtrcII` MCVB matrix generation. The current nonorthogonal-orbital CI matrix generator, this supports discrete and continuous point groups.

- CI eigensolution modules

  `neweig` Small CI matrix diagonalization. Produces a requested number of lowest energy eigenvalues and eigenvectors. Not suitable for large matrices but capable of delivering all roots.

  `beigII` Large CI matrix diagonalization. Produces a small number of the lowest energy eigensolutions, but is suitable for larger matrices.

- Parameter-optimization module

  `smplxtrdg` Simplex optimization. A simple but robust optimizer often used for geometry optimization, but capable of optimizing any input parameter.

Secondary Modules:

Modules which calculate properties or interpret the results of the principal CRUNCH modules can be divided into those concerned with orbitals, those which deal with SCF wavefunctions, and those which follow CI calculations. More extensive documentation for these modules will be included in later versions of this document. Meanwhile, consult the usage statements printed when the module is invoked without additional parameters.

- Orbital property modules

  `ptgorbv` Orbital value, Gaussian basis. Calculates the value, at a point, of a single Hartree-Fock orbital in a Gaussian basis.

  `ptsorbv` Orbital value, Slater basis. The same as `ptgorbv`, except for a primitive Slater orbital.

  `mpole` Orbital multipoles. This module, not generally run independently, is automatically called before several of the multi-electron property modules to supply the one-electron, orbital-multipole integrals.

  `mcdip` Orbital multipoles. This is an alternative to `mpole` used (automatically) by several of the multi-electron property modules.

- SCF property modules

  `ptgpot` Potential, Gaussian basis. Gives at a specified point the electrostatic potential of an R(O)HF wave function composed of Gaussian orbitals.

  `ptspot` Potential, Slater basis. The same as `ptgpot`, except for a wave function in a Slater basis.

  `ptscrgden` Density, Slater basis. Gives at a specified point the charge density of an R(O)HF function in a Slater basis.

  `dipole` R(O)HF multipole moments. Although a separate module (which the user can call directly), this is called automatically after the primary module, `gscfnn`, to provide static multipole moments for R(O)HF functions.

  `udipole` UHF multipole moments. The same as `dipole`, but for UHF functions.

- CI property modules

  `vbmpole` CI multipoles. Calculates electrostatic multipole moments of either a nonorthogonal or orthogonal CI wavefunction by a more rapid direct method.

  `vbdnsty` CI multipoles via the density matrix. Calculates the first-order, spin-free density matrix for a CI wave function and then uses this to obtain the natural orbitals and the electrostatic moments.

  `tranmom` CI transition moments. Calculates transition dipoles between orthogonal or nonorthogonal CI states.

  `egsopop` EGSO population. Computes the *eigenvector-guided sequential orthogonalized* population analysis (see [3], Sect. 1.4) for the results of either a nonorthogonal or orthogonal CI calculation.

  `hlspvb` HLSP decomposition. Transforms a CI eigenvector into Heitler–London–Slater–Pauling perfect-pairing functions (see [4]).

Support Modules:

The final class of modules are those which provide support functions such as input preparation, output manipulation, interface to other suites, and other stand-alone features. These will also have more extensive documentation in later editions of this document. Users are again referred to the usage statements.

- Input preparation modules

  `mkscf` SCF input preparation. An aid to creating input for the various Hartree-Fock programs especially useful for specifying orbital symmetry.

  `mklpscf` SCF input preparation for linears. A similar module for calculations in the $C_{\infty v}$ and $D_{\infty h}$ point groups.

  `synp` Input preparation for `symgenn`. Helps to produce the generalized permutation table which specifies orbital symmetry in the configuration generator.

`symgerr` Error detection for `symgenn`. Run automatically after the configuration generator, this checks for errors in the configuration list, allowing the input to be corrected without wasting time generating a defective CI matrix.

- Output manipulation modules

  `get-ao-tran` AO transformation table. Extracts from the `lobeWat` output the table giving the transformation of the AO basis under the operations of the point group. Useful for preparing input to `mkscf`.

  `get-e-order` Energy order of R(O)HF orbitals. Extracts the global energy order of the orbitals (and corresponding symmetry labels) which are otherwise ordered by energy within symmetry species and displays them in a convenient format.

  `prntmos` R(O)HF orbitals. Reads a binary molecular-orbital file and displays its contents in a convenient format, either ordered by orbital energy or symmetry species. Called automatically after `mxsscf`.

  `prths` H and S matrices. Extracts from their binary files the Hamiltonian matrix and, if present, the overlap matrix for a CI problem.

  `prthsv` H, S, and eigenvector matrices. Same as the previous, but also prints the eigenvector matrix.

  `stats` Problem statistics. Prints selected information from outputs sharing a file-name prefix.

  `bigtab` Tabular eigensolutions. Sorts through a CI vector produced by `beigII` and displays the largest contributions in a convenient form.

  `fndsymfun` CI symmetry functions. Extracts from the list produced by `symgenn` the symmetry function specified by the user and displays it in a convenient form.

- Intersuite communication modules

  `zmat` Z-matrix conversion. Converts a molecular geometry from Z–matrix form (see [5], sect. 3.3) into Cartesian coordinates for `lobeWat` input.

  `gmscnv` Orbital conversion from GAMESS. Reads the MOs produced in an R(O)HF calculation with the GAMESS[6] suite and writes them into a binary file readable by CRUNCH.

- Miscellaneous utility programs and scripts

  `cmpt` Compare times. Warns if input files are newer than output files.

  `lnkfls` Link files. Makes symbolic links so that a single AO integral set can be used for multiple calculations with different file-name prefixes. Saves the space required by copies and the time consumed in recalculation.

  `modfls` Modify files. Renames a complete set of input files to a new file-name prefix. Useful for starting a new problem similar to an existing one.

3 General Comments on Usage and Files

Although referred to in this manual generically as "modules," the computational pieces of the CRUNCH suite vary dramatically in size and complexity. In fact, some are small shell scripts and even some of the compiled codes, especially those outside of CRUNCH's core capabilities, are expected to be called directly from the command line. Typing the name of the executable with no additional parameters produces a short usage statement.

Although all the principal modules can be invoked in the same way, those which are commonly used successively to perform common types of calculations can also be invoked in order with the executable `crunch` script. Its usage statement describes the modules supported.

```
Usage: crunch -[ciahxujtCsSPoOXmMebvVdDTRz] <prefix>
-c  Cleans intermediate and output files from working directory
-i  Lobe Gaussian integrals
-a  Atomic Slater integrals
-h  SCF[R(O)HF] calculation
```

```
-x  Maximum-spin R(O)HF calculation
-u  UHF calculation
-j  Orbital matrix meld
-t  Transform integrals
-C  Complex integral transformation for linear pt. grps.
-s  Symgen for discrete point groups
-S  Symgen for linear point groups
-P  MP2 calculation
-o  Orthogonal-orbital CI-matrix generation, discrete pt. grps.
-O  Orthogonal matrix generation for linear pt. grps.
-X  MCSCF calculation
-m  Nonorthogonal matrix generation, discrete pt. grps.
-M  Nonorthogonal matrix generation for linears
-e  Eigenvalues of small CI matrices
-b  Eigenvalues of CI matrix by large matrix method
-v  Heitler-London-Slater-Pauling weights of a CI vector
-V  EGSO analysis of a CI vector
-d  Faster method for electric moments of a CI vector
-D  Natural orbitals and moments of a CI vector via density matrix
-T  Dipole transition moments for CI vectors
-R  Runs a reader on .out files in reverse order
-z  Converts Z-matrix to Cartesian geometry
```

The `-c` option deserves special comment. The first step in diagnosing and remedying errors or unexpected behavior is beginning again with a fresh start; clean early and often.

Note also that the options are intended to be concatenated. For example, assuming that all of the appropriate input files (whose names begin with `hf`) had been constructed for the HF molecule, the command:

```
crunch -ihtsme hf
```

would be used to perform a nonorthogonal CI calculation, starting with the Gaussian integrals and finishing with the matrix eigensolution. One must be familiar with the theory of such calculations to know the proper order in which the programs must be run. (Although, more often than not, it is the order in which the modules are described in this manual.) Examples of a large number of *ab initio* valence bond calculations are given in a recent book.[3] In addition, the examples provided with CRUNCH exercise each of the modules and illustrate many of the types of calculations which are possible by employing the modules successively.

The CRUNCH modules communicate with one another through disk files, and each has one or more input files and one or more output files. Some of these are plain text files and some are binary; some are user-supplied and some are automatically generated for intermodule communication; some are optional and some are required. Nearly all of the disk files provided or produced have file names with a particular structure (exceptions are noted later), having a *prefix*, a *suffix*, and an *extension*.

prefix This is a set of letters or characters of the user's choice that identify the problem. All of the characters must be taken from the set of legal UNIX® file-name characters. The prefix may end in a period, which, however, is considered part of the prefix.

suffix This is a set of three or more letters unique to the module producing it or the information stored in it. These are predetermined in the programs and must be adhered to.

extension This is a set of one or more letters or numbers following a required period. For those modules requiring a user constructed input file, the extension is generally `.inp`.

As an example, we note that the suffix for the first module commonly run is `lob` and it requires a user-constructed input file and produces a text-format output file. Therefore, if we were making calculations of the molecule HF, an acceptable file name for the input to this program would be `hflob.inp` and the module would create a `hflob.out` file.

In the sections of this manual describing each module in detail, tables are given with the file names and a brief description of their contents. Primarily a reference guide to insure that the user correctly supplies the required input files, these tables may also aid in discerning the order in which the modules are to be invoked.

Comments on Input File Format:

The user-constructed files can be produced by any editor that adheres to the UNIX® convention for a new line. With a few exceptions, only user-constructed input files have the `.inp` extension, so if we speak of a `.inp` file below (or, *e.g.* a `lob.inp` file or `<prefix>lob.inp` file), the reader may generally assume it is to be produced by hand with an editor. CRUNCH has relatively few built-in defaults. Again, this provides maximum flexibility. The disadvantage, of course, is that it also provides considerable scope for errors in the input. Not only can these lead to incorrect results, but segmentation faults or other system malbehavior may even occur.

The input files we have been describing allow the inclusion of comments. These all start with either a `#` or a `;` character, which may be placed at any position in the line, and continue until the end of the line. The comments are allowed anywhere with a few exceptions, where they cannot appear before a required title at the beginning of the dataset. In following sections the instances of this restriction are noted. Unless otherwise stated, all numbers and strings in the input are in free format, with spaces or new-lines separating them. (When maximum string lengths are not specified, the underlying data structures have been dimensioned to sizes well in excess of what is expected to be necessary.) As with the input to all C programs, white-space separated data in a prescribed order can have new-lines arbitrarily interspersed without ill effect. Nevertheless, clarity suggests that certain data lines not be split. In the following module-specific descriptions of the input-file format, sensible groupings are given. In addition, constraints on format and range of values are specified. Unless otherwise noted, all positions, distances, and energies are to be given in atomic units.

A list of the principal input files required by each module is given for reference in Table 1. (Modules missing from the table require no input file. A few additional, optional input files (generally used to access special, less common features) are described later.)

Table 1: The principal input files read by CRUNCH.

Module	Suffix.Extension
`lobeWat`	`lob.inp`
`atmintc`	`atm.inp`
`gscfnn`	`scf.inp`
`mxsscf`	`scfx.inp`
`uhfnn`	`uhf.inp`
`matmld`	`mld.inp`
`trannn`	`trn.inp`
`ctran`	`ctrn.inp`
`symgenn`	`sym.inp`
`mp2`	`mp2.inp`
`mcscf`	`mchf.inp`
`beigII`	`big.inp`
`smplxtrdg`	`plx.inp`

This manual continues with an important and general discussion of the use of symmetry in CRUNCH and its specification in the input files. Impatient users are warned against neglecting it. Following this, we discuss in more detail each primary module with particular emphasis on the preparation of correct input datasets.

4 Symmetry in CRUNCH

The use of symmetry can, of course, improve the efficiency of molecular and atomic calculations dramatically. The description of symmetry and group theory as it impacts electronic structure calculations is a voluminous topic and, of necessity, only a brief description can be given in this manual. Nevertheless, even here, considerable space is required to give a detailed description of the use of point group theory in the CRUNCH modules.

Those familiar with other quantum chemical packages will note that many restrict their automatic symmetry analysis to Abelian groups, *i.e.* those groups for which all operations commute in pairs. CRUNCH is more flexible, at some cost of automation and ease of use, and the different modules vary considerably in their abilities to deal with non-Abelian groups. However, there is normally no reason that an appropriate subgroup cannot be used at each stage of a sequence of CRUNCH calculations in case comparison with other code suites is desired.

At primary issue here is the description of spatial symmetry groups in terms of their generalized permutation representations. In the next subsection we define these, and in the following subsections we describe how symmetry impacts modules in four different categories: integral generation, R(O)HF and UHF calculation, integral transformation, and MCSCF and CI calculation. Further details for each of the specific modules will be taken up separately when their input datasets are described; the present discussion will cover background only. In addition, the initial discussion will focus on discrete symmetry point groups. (The special considerations required for the continuous groups, $C_{\infty v}$ and $D_{\infty h}$, are best understood as modifications of the framework of the discrete cases and so their discussion is somewhat delayed in the following.)

4.1 Generalized Permutation Representations

The concept of a generalized permutation representation is important for understanding the operation of CRUNCH. This has been detailed in a discussion of the symmetry of n-electron functions[7] and, here, only a brief outline is given.

A *permutation matrix* is a square matrix in which each column and row has one and only one nonzero element and the value of that element is restricted to 1. It follows directly from the definition of a group that each finite group possesses at least one representation made up entirely of permutation matrices.[1]

If now we generalize this idea slightly and allow the single nonzero element in each row and column to be of unit magnitude, we can call this a *generalized permutation matrix.* As CRUNCH does not currently support the use of complex numbers in this connection, the nonzero entries of the generalized permutation matrices of present interest (the *real generalized permutation matrices*) are limited to ± 1. We give an example in the next section.

4.2 Symmetry and Integral Generation

The integral packages differ in the way in which they employ spatial symmetry.

- `lobeWat`

  The efficiency of the Gaussian integral module depends significantly upon the use of spatial symmetry. Let us consider a normalized atomic orbital basis, $\chi_1, \chi_2, \ldots, \chi_n$. If the molecule possesses elements of symmetry, the basis should reflect these as:

$$\hat{R}\,\chi_i = \sum_j \chi_j\, D(\hat{R})_{ji},$$

  where $\hat{R}$ is a symmetry operation, $D(\hat{R})_{ji}$ are the elements of the representation matrix, and the basis must be closed under the operations of the group. If the $\mathbf{D}(R)$ matrices are all of the real-generalized-permutation type, then the full symmetry of the molecule may be utilized to speed the production of the integrals. Whether this is possible, or whether a subgroup must be used instead, depends upon the interplay of several factors: the nature of the system, the properties of its spatial group, the basis one wishes to use, and, perhaps surprisingly, the fact that there are three spatial dimensions. It seems most illuminating to give a simple example where the C_{3v} group supports one such basis and to describe further circumstances where a subgroup must be used instead.

  Consider a minimal basis calculation of the NH_3 molecule with a geometry satisfying C_{3v} symmetry. Further, use a reference frame in which the Cartesian coordinates of the nuclei are as shown in Table 2. The basis is ordered as $\{N\,1s,\ N\,2s,\ N\,2p_1,\ N\,2p_2,\ N\,2p_3,\ H_1 1s,\ H_2 1s,\ H_3 1s\}$ with the three orthogonal

[1] A finite group possesses a square *multiplication table*, each row of which is merely a permutation of the row above. If these permutations are converted into permutation matrices, they collectively constitute a permutation representation of the group.

2p orbitals oriented trigonally and the subscript indicating to which hydrogen each is (only nominally) directed. Therefore, the unit vector perpendicular to the nodal plane of N 2p_1 is in the yz-plane at an angle of $\arccos(1/\sqrt{3})$ ($\approx 54.7^o$) from the z–axis.[2] (Note that this angle is independent of the molecular bond angles.) The other nitrogen 2p orbitals are arranged to have the same projection on the z-axis. The projections on the xy-plane of the lines perpendicular to the nodal planes of these orbitals are at an angle $2\pi/3$ (120^o) from the y-axis and are contained in the other two vertical symmetry planes (labeled σ_{120} and σ_{240} below).

Table 2: Atom positions for ammonia.

Atom	x	y	z
N	0	0	0
H	0	a	b
H	$-\sqrt{3}a/2$	$-a/2$	b
H	$\sqrt{3}a/2$	$-a/2$	b

* a and b are not determined by symmetry, but by bond distances and angles

The permutation matrix resulting from subjecting this ordered orbital basis to the C_3 operation is:

$$\mathbf{D}(C_3) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and one can begin to see how a permutation representation generalized to include elements of -1 might become necessary in other circumstances, such as in point groups with a center of symmetry or when orbital nodal planes coincide with symmetry planes.

The results of subjecting the entire basis to all the operations of the point group are shown in Table 3 and the representation of C_{3v} supported by this basis is of the type we have been discussing.

Table 3: Transformation of a specially oriented basis on ammonia.

I	C_3	C_3^2	σ_{yz}	σ_{120}	σ_{240}
N 1s	N 1s	N 1s	N 1s	N 1s	N 1s
N 2s	N 2s	N 2s	N 2s	N 2s	N 2s
N 2p_1	N 2p_2	N 2p_3	N 2p_1	N 2p_3	N 2p_2
N 2p_2	N 2p_3	N 2p_1	N 2p_3	N 2p_2	N 2p_1
N 2p_3	N 2p_1	N 2p_2	N 2p_2	N 2p_1	N 2p_3
H_1 1s	H_2 1s	H_3 1s	H_1 1s	H_3 1s	H_2 1s
H_2 1s	H_3 1s	H_1 1s	H_3 1s	H_2 1s	H_1 1s
H_3 1s	H_1 1s	H_2 1s	H_2 1s	H_1 1s	H_3 1s

Of course, in the coordinate frame given in Table 2 the p-basis could have been directed along the Cartesian axes and the molecule treated in C_s symmetry as is commonly done in some quantum chemistry codes. In that case, degeneracies of orbitals and states would not be so precise.

The situation becomes more nuanced if the basis includes d-orbitals. The spherical set, $d_{2z^2-x^2-y^2}$, $d_{x^2-y^2}$, d_{zx}, d_{zy}, d_{xy}, cannot be arranged trigonally to produce a generalized permutation representation, but the Cartesian set, d_{x^2}, d_{y^2}, d_{z^2}, d_{xy}, d_{xz}, d_{yz}, can be. Such arrangements are possible for basis sets proportional to products of Cartesian coordinates (and the capabilities of the `lobeWat` module in this regard will be discussed later.)

[2] As we shall see, arbitrary orientation of the orbitals is allowed by `lobeWat`.

Further, it is worth noting those circumstances under which the real generalized permutation representation of CRUNCH is not sufficient to encompass a symmetry group. In particular, if there are atoms on higher-order axes, some rotations may transform a basis orbital into a combination of more than one of the basis orbitals. In this case, the largest subgroup supporting the required representation may be employed. (Thus, atoms described by a Cartesian basis may be treated using D_{4h}.) On the other hand, benzene provides a good example of the use of higher point groups for molecules lacking problematic atoms. In a D_{6h} geometry, we could orient atom-centered Cartesian systems so that each has its z-axis perpendicular to the plane of the molecule. Then each of the local x-axes could be oriented radially (*e.g.* outward) and the y-axes could be directed tangentially with a common vorticity. These axes could then be decorated with either spherical or Cartesian atomic orbitals in such a way as to support the required representation.

- `atmintc`

  The Slater integral package has no symmetry group requirements that the user need be aware of. More will be said about its interface to the Hartree-Fock modules in the next section.

4.3 Symmetry and the R(O)HF and UHF Modules

The degree of symmetry allowed by these modules depends upon the source of the integrals.

- `lobeWat`

  Using the more-established methods for discrete point groups, the level of symmetry allowed in Hartree-Fock calculation is the same as that in integral generation. However, a new feature arises. For a degenerate irreducible representation (IR), the orientation of the components must be specified. (In this manual, when these components need to be distinguished from the irreducible representations themselves, they are referred to as *symmetry species.*) Although this choice, global to the final molecular orbital basis, can be done manually, the `mkscf` program is provided as an aid. The `lobeWat` output contains a table of the AO transformations for the generalized permutation representation. This table may be extracted with the `get-ao-tran` utility and pasted into the input file for `mkscf`.

  To take advantage of CRUNCH's newer capability to use the continuous linear point groups, $C_{\infty v}$ and $D_{\infty h}$, the proper symmetry species for atoms or diatomics can be constructed, with some manual intervention, using the `mklpscf` module.

- `atmintc`

  The atomic Slater integral package provides integrals which can be used by the Hartree-Fock modules to perform the calculation in full symmetry. `atmintc` has two operating modes, with and without an applied electric field along the z-axis. In the absence of a field, full spherical symmetry can be employed.[3] With the field the system symmetry is $C_{\infty v}$. The output file from `atmintc` contains a template for the input to the Hartree-Fock programs which automatically specifies the symmetry species of the orbitals in either case.

  This full-symmetry treatment of atoms is not maintained throughout CRUNCH. Post-Hartree-Fock calculations must retreat at least to $D_{\infty h}$ if the somewhat new features for the linear continuous groups are tried or to D_{4h} if constrained to the more-established methods.

Regardless of the integral package used, when symmetry is specified, the Hartree-Fock modules can determine the lowest-energy state of any requested symmetry species, even if it does not contain the system's ground state. In addition, `gscfnn` can, in some circumstances, be used to determine state-averaged solutions among several symmetry species or even irreducible representations.

4.4 Symmetry and Orbital Transformation

The transformation of the one- and two-electron integrals corresponding to an orbital basis change generally takes place in one, and sometimes a second, step.

[3]This is true with the caveat that the solutions obtained are actually the real linear combinations of the familiar complex eigenfunctions of good angular momentum (in particular, the symmetric and antisymmetric combinations of functions with the same $|m|$ in O(3)). These multi-electron functions have the same inter-relationship as the real and complex spherical harmonics. (The Hartree-Fock solutions for the linear continuous point groups are similarly real.)

- `trannn`

  This module transforms the integrals produced by `lobeWat` or `atmintc` in an atomic-orbital basis into integrals in a Hartree-Fock basis (or in some other, user-specified basis). In this module no symmetry information need be specified, none is used to speed the calculation, and none is explicitly recorded in the transformed integrals.

  There is a core-valence separation allowed and, when used, the core[4] is commonly composed of orbitals belonging to definite symmetry species.

- `ctran`

  This module is used only for systems with $D_{\infty h}$ or $C_{\infty v}$ symmetry. Its exclusive purpose is to transform the integrals to reflect a conversion of the underlying basis from real functions into complex combinations of these functions, especially those that are eigenfunctions of the $\hat{L}_z$ operator. Although not necessarily indicative of the actual algorithms employed or the data stored, the one-electron basis can thereafter be regarded as complex.

4.5 Symmetry and the CI Modules

All of the CI methods in CRUNCH, the MRCI, the MCVB, and the MCSCF, depend upon the `symgenn` module for configuration selection and the operation of this module is somewhat different for finite point groups and for the continuous linear groups.

The implications of the restriction to a real generalized permutation representation of the orbitals are especially apparent for nonlinear molecules. Although there are exceptions, one often performs CI calculations with orthogonal molecular orbitals that form a basis for a completely reduced representation of the spatial group. Since the number of groups that possess such irreducible representations is small, relatively few such groups can be used at this level. The finite groups for which this is possible have at most two- or four-fold rotation axes. Thus, for example, a minimal basis full-π MRCI calculation of the benzene molecule (in a D_{6h} geometry) would be limited to the D_{2h} point group.

The exceptions are mainly MCVB calculations where localized atomic orbitals are used directly in configurations. Such a calculation of benzene can be performed in D_{6h} and provides a multi-electron counterpart to the benzene example discussed above in connection with integral generation.

For atoms or linear molecules with complex orbitals, `symgenn` combines a real generalized permutation representation of the orbitals over a subset of the group operations with a partial accounting of angular momentum symmetry (see the `symgenn` section for details) to treat $D_{\infty h}$ or $C_{\infty v}$ systems explicitly. Unlike the full-symmetry linear or atomic Hartree-Fock solutions, the multi-electron wavefunctions which result can be thought of as being complex (although this doesn't characterize their computational representation). They have a relationship to corresponding real functions analogous to the relationship between spherical harmonics and their real counterparts. CRUNCH is not currently capable of performing atomic CI calculations with full spherical symmetry.

We continue now with the bulk of the manual which is a discussion of each primary module and emphasizes the details of constructing the user-supplied input files.

5 Primary Computational Modules

5.1 Integral Generation

5.1.1 `lobeWat`

This module calculates the one- and two-electron integrals for atoms and molecules from Cartesian or spherical lobe-Gaussian[8] atomic orbitals (AOs). The input is only slightly different from what would be necessary if standard Gaussians were used. Table 4 contains a list of the files read or written by this module.

[4]The somewhat unconventional use of this term is more fully explained in the `trannn` section.

Table 4: Files read (above the single line) or written (below) by `lobeWat`

Suffix and Extension or (Full File Name)	Description (opt.=optional)	Text (t) or Binary (b)
`lob.inp`	User-supplied input	t
(User specified)	Nuclear charges and positions (opt.)	t
`lob.out`	Output file	t
`enuc.dat`	Nuclear Energy	b
`aop.dat`	AO basis statistical info.	t
`lob.I1`	One-electron integrals	b
`lob.I2`	Two-electron integrals	b
`lob.I2n`	Two-electron integrals info.	t

Deferring specification of its location and local orientation, a single, primitive Gaussian function can be written as:

$$g_{\mathrm{LP}\alpha}(\overrightarrow{r}) = N_{\mathrm{L}\alpha}\, r^{\mathrm{L}}\, e^{-\alpha r^2}\, y_{\mathrm{LP}}(\hat{r})$$

where $N_{\mathrm{L}\alpha}$ ensures normalization and, modifying the isotropic polynomial and exponential terms, is the orientation term, $y_{\mathrm{LP}}(\hat{r})$. For spherical Gaussians these are just the real spherical harmonics, $Y_{\mathrm{LP}}(\theta, \phi)$, and L and P are the familiar quantum numbers l and m. For these spherical functions, CRUNCH supports angular momentum up through L=2 (d orbitals). For Cartesian functions the $y_{\mathrm{LP}}(\hat{r})$ are the unique products (normalized to r^L) of the Cartesian coordinates with combined exponents equal to L (a generally enlarged set with respect to spherical functions of the same L).[5] For Cartesian functions the maximum value of L supported here is three (f orbitals).

As is common in Gaussian quantum-chemical codes, primitive functions with the same L and P but different α can be combined in a fixed linear combination as:

$$G_{\mathrm{LP}}(\overrightarrow{r}) = y_{\mathrm{LP}}(\hat{r})\, r^{\mathrm{L}} \sum_i C_i\, e^{-\alpha_i r^2}$$

to produce a single normalized function with tunable r-dependence to more closely approximate a Slater function or otherwise produce an optimal shape. As an aid to optimizing or adapting existing bases, `lobeWat` allows all the α in each combination to be multiplied by a single, fixed scale factor. These combined functions are also termed *group functions* below and the coefficients are commonly referred to as *contraction coefficients*. Strictly speaking, as used by `lobeWat`, both the primitive and the combined functions should be distinguished from "atomic orbitals," the main difference being that by being placed at several locations with definite orientations, a single group function can be used to define multiple atomic orbitals.

It is worth emphasizing that, not only must the orientation of each atomic orbital be specified by the user, but it may also be positioned arbitrarily, not just at the position of a nucleus. In the interests of flexibility, these positions may be optionally read from a separate file. A single sphere of constant charge (a Watson sphere[10]) may also be placed around the system.

Format of the `<prefix>lob.inp` File:

The input dataset may have comments throughout but the *first two* lines of the input are printed for reference. It is good practice to use them for comments to identify the problem.

The input data are divided into the sections given below. Line numbers are internal to these sections.

- General statistical information

  Line 1: nuc norb ntyp ngrp I1flg I2flg

nuc	number of nuclei and AO positions (int)
	If nuc < 0 the nuclear positions are given in a separate file (see below).
norb	number of atomic orbitals (int)
ntyp	number of group functions (int)

[5]In general, this set of Cartesian functions of fixed L not only spans the complete set of spherical harmonics with L=l, but may also include spherical harmonics of lower l. Nevertheless, this manual accords with common usage[9] in, for example, referring to the six Cartesian Gaussians with L=2 as "d-functions" even though they include an s-function.

ngrp	number of spatial symmetry group elements (int)
I1flg	'1' - calculate one-electron integrals '0' - do not calculate
I2flg	'1' - calculate two-electron integrals '0' - do not calculate

- Geometric scale factor and Watson sphere

  Line 1: gscal

gscal	multiplicative scale factor for nuclear and AO positions and displacements and the Watson sphere size and position (float)

  Line 2: wq wrad wcx wcy wcz

wq	charge on Watson sphere, in au (float)
wrad	radius of Watson sphere (float)
wcx	x position of center of Watson sphere (float)
wcy	y position of center of Watson sphere (float)
wcz	z position of center of Watson sphere (float)

- Nuclear and/or atomic orbital positions

  There are two possibilities here. The charges and positions may be included in the input file or read from an external file (if nuc < 0 above). In either case the following format for the charges and positions is used. When reading from an external file there is an additional option useful for moving some of the centers in one portion of a molecule (listed first among the positions) with respect to the remainder.

  If nuc > 0:

  nuc lines: zz rx ry rz

zz	nuclear charge (float, note: may be nonintegral) Set to zero for the position of an AO not on a nucleus.
rx	x-coordinate of position (float)
ry	y-coordinate of position (float)
rz	z-coordinate of position (float)

  Or if nuc < 0:

  Line 1: fname

fname	name of external file with nuclear and AO positions (string)

  ***** Beginning of the format of the fname file**

  First abs(nuc) lines of file fname: zz rx ry rz

  (as above for nuc > 0)

  Next line of file fname: nvec

nvec	number of the initial centers in the list to translate (int, $0 \leq$ nvec $\leq$ nuc) If nothing is translated, this must still be present and $= 0$.

  Next nvec lines of file fname: tx ty tz

tx	x-displacement of the entered position (float)
ty	y-displacement of the entered position (float)
tz	z-displacement of the entered position (float)

  ***** End of the format of the fname file**

- Atomic orbitals – types and orientations

  Each of the norb lines of this section specifies a single combined orbital (the combinations are given in the next section). The number of items required on each line depends upon the type of the orbital (*i.e.* its angular momentum, orientation, and whether it is Cartesian or spherical). For clarity the first five required parameters common to all orbital types are described first, then the particular cases which define the orientation of anisotropic orbitals and necessitate up to six additional parameters *on the same line* are considered separately.

  First norb lines: labl L ltyp otyp opos

labl	arbitrary alphanumeric AO label (string, 1-10 char.)
L	value of L in the second equation of this section (int = 0, 1, 2, or 3)
ltyp	orbital type (int = 0, 1, 2, 3, 4, or 5)

For an s-orbital (L = 0):
ltyp = 0

For a p-orbital (L = 1):
ltyp = 0

For a d-orbital (L = 2):

ltyp = 0	spherical d_{z^2}-orbital
ltyp = 1	one of the other four spherical d-orbitals
ltyp = 2	Cartesian d_{xx}, d_{yy}, or d_{zz} orbital
ltyp = 3	Cartesian d_{xy}, d_{xz}, or d_{yz} orbital

For an f-orbital (L = 3):

ltyp = 3	Cartesian f_{xxx}, f_{yyy}, or f_{zzz} orbital
ltyp = 4	Cartesian f_{xxy}, f_{xxz}, f_{yyx}, f_{yyz}, f_{zzx}, or f_{zzy} orbital
ltyp = 5	Cartesian f_{xyz} orbital

(ltyp = 0, 1, or 2 reserved for spherical orbitals, but not implemented; only Cartesian f-orbitals are currently available.)

otyp	number in the list of ntyp group functions (next section) for this AO (int)
opos	location of AO as number in the list of abs(nuc) positions (int)

Polar angles possibly appended to each of the **first norb lines** (all float):

For s-orbitals (L = 0): (no additional parameters)

For p-orbitals (L = 1): theta1 phi1

theta1	azimuthal angle of positive lobe
phi1	equatorial angle of positive lobe

For spherical d-orbitals (L = 2 and ltyp ≤ 1):

If ltyp = 0, a d_{z^2} orbital: theta1 phi1

theta1	azimuthal angle of either positive lobe
phi1	equatorial angle of that positive lobe

If ltyp = 1, one of the other four spherical d orbitals: theta1 phi1 theta2 phi2

theta1	azimuthal angle of either positive lobe
phi1	equatorial angle of that positive lobe
theta2	azimuthal angle of either negative lobe
phi2	equatorial angle of that negative lobe

For Cartesian d-orbitals (L = 2 and ltyp ≥ 2):

If ltyp = 2, a d_{pp} orbital (p = x, y, or z): theta1 phi1

theta1	azimuthal angle of either positive lobe
phi1	equatorial angle of that positive lobe

If ltyp = 3, a d_{pq} orbital (pq = xy, xz, or yz): theta1 phi1 theta2 phi2

theta1	azimuthal angle of the p direction
phi1	equatorial angle of the p direction
theta2	azimuthal angle of the q direction
phi2	equatorial angle of the q direction

For Cartesian f-orbitals (L = 3 and ltyp ≥ 3):

If ltyp = 3, a d_{ppp} orbital: theta1 phi1

theta1	azimuthal angle of the positive lobe
phi1	equatorial angle of that lobe

If ltyp = 4, a d_{ppq} orbital: theta1 phi1 theta2 phi2

theta1	azimuthal angle of the p direction
phi1	equatorial angle of the p direction
theta2	azimuthal angle of the q direction
phi2	equatorial angle of the q direction

If ltyp = 5, a d_{xyz} orbital: theta1 phi1 theta2 phi2 theta3 phi3

theta1	azimuthal angle of the x direction

phi1	equatorial angle of the x direction
theta2	azimuthal angle of the y direction
phi2	equatorial angle of the y direction
theta3	azimuthal angle of the z direction
phi3	equatorial angle of the z direction

- Group/combination functions

  ntyp groups of lines: Each group defines one of the group functions.

  First line of group: typn scale

typn	number of primitive Gaussians in combined function (int) Set to one to use a primitive directly.
scale	multiplicative scale factor for all alf (below) in group (float)

  Next typn lines of group: alf coef

alf	exponential scale factor of primitive in group function (float)
coef	contraction coefficient of primitive in group function (float)

- Spatial symmetry group elements.

  ngrp lines: gop+gopord goppwr theta phi

gop	operation type (char) 'C' - proper rotation 'S' - improper rotary reflection
gopord	order of the rotation axis (int) (*i.e.* the value for which 2π/gopord is the rotation angle)
goppwr	the exponent which produces the operation (int) (*e.g.* 2 in C_3^2)
theta	polar azimuthal angle of the axis of rotation (float)
phi	polar equatorial angle of this axis (float)

  The first two parameters are written together with no intervening spaces.
  Note that, although formally only proper and improper rotations are supported, 'S1 1' gives a standard reflection operation, 'S2 1' the inversion, and 'C1 1' the identity. Also, for example, in a group with a C_4 axis, the composite C_4^2 operation can be entered as such ('C4 2') or as C_2 ('C2 1')) with equal validity.

5.1.2 `atmintc`

This program calculates the one- and two-electron integrals for single atoms from real, Slater orbitals. Table 5 contains a list of the files read or written by this module.

The format and naming of the one- and two-electron integral files is the same as that used for the lobe Gaussians which are, in some sense, the CRUNCH default; it is up to the user to keep track of the fact that the integrals here are not produced by the `lobeWat` program. The AO dipole transition moments and the spatial part of the one-electron spin-orbit coupling matrix elements are also calculated. Automatically appended to the output file is a template for the input file for the following single-configuration SCF modules. It is also possible to perform the calculation with a finite (*i.e.* noninfinitessimal or nonperturbative) electric field applied along the z-axis.

This module employs Slater orbitals as defined by:

$$S_{\mathrm{nlm}\alpha}(r,\theta,\phi) = N_{\mathrm{n}\alpha}\, r^{\mathrm{n}-1}\, Y_{\mathrm{lm}}(\theta,\phi)\, e^{-\alpha r}$$

where $N_{\mathrm{n}\alpha}$ is the normalization constant and the normal restriction on quantum numbers, $\mathrm{n} \geq \mathrm{l}+1$, is enforced, but there is no limit imposed on the maximum value of n. The input data require only the unique sets of n, l, and α. The program expands this set into the full number, 2l+1, accounting for the different m-values. The real spherical harmonics, $Y_{\mathrm{lm}}(\theta,\phi)$, have been verified to obey the phase and direction conventions of reference [12].

Format of the `<prefix>atm.inp` File:

After any initial comment-only lines, the following data are given (opt. = optional):

Line 1: norbc Z

Table 5: Files read (above) or written (below) by `atmintc`

Suffix and Extension	Description	Text (t) or Binary (b)
`atm.inp`	User-supplied input	t
`atm.out`	Output file	t
`enuc.dat`	Nuclear Energy	b
`aop.dat`	AO basis statistical info.	t
`lob.I1`	One-electron integrals	b
`lob.I2`	Two-electron integrals	b
`lob.I2n`	Two-electron integrals info.	t
`atmom.I1`	Transition moment integrals	b
`lob.SOI1`	Spin-orbit coupling matrix elements	b

norbc	number of sets of n, l, and α values (int)
Z	nuclear charge (float, note: may be nonintegral)

Line 2: print_flag

print_flag	'0' gives minimal printout
	'1' prints the nonzero one- and two-electron integrals to the output file

Line 3 (opt.): label value

label	The string: 'FF'
value	electric field strength (in au) (float)

If this entire line is missing there is no applied field.

Next norbc lines: n l α

n	principal quantum number of the orbitals (int)
l	azimuthal quantum number of the orbitals (int, l<n)
α	exponential scale factor of the orbitals (float)

5.2 Hartree-Fock Calculation

5.2.1 `gscfnn`

This program finds the spin-restricted, open- or closed-shell, self-consistent-field (SCF) solutions of the Hartree-Fock Hamiltonian (abbreviated as R(O)HF). It does this for a fixed configuration of the nuclei using the atomic orbital (AO) integrals generated by either `lobeWat` or `atmintc`. Table 6 contains a list of the files accessed by this module.

For the most general spin case, the eigenvalues are given in terms of the one-electron energies and the two-electron Coulomb and exchange matrix elements as:

$$E = \sum_i \mathrm{W}_i h_{ii} + \sum_{ij} (\alpha_{ij} J_{ij} - \beta_{ij} K_{ij})$$

where the sums are over the molecular orbitals (MOs) determined by self consistency. The theory behind the algorithm, in particular the meaning of the spin-coupling constants, W, α, and β, is given by Carbo and Riera.[14] For many common circumstances these parameters are quite simple, but when they are appropriately constructed, quite sophisticated calculations are possible. See the Appendix of this user's manual for further discussion of these parameters along with several examples, some simple and a few more complex.

It should be noted that the number of electrons and the spin of the overall wave function are not specified directly by the user but are derived from the spin-coupling parameters and the occupations of the orbitals. The user is admonished to check that the number of electrons and the overall spin given in the output are as intended, as there is no internal check of these values. Even calculations with parameters implying a fractional number of electrons will quietly attempt to proceed to a meaningless result.

In addition, the user must explicitly symmetry adapt the AO basis for each symmetry species. (Revisit section 4.3 for a discussion of a symmetry species.) Each basis orbital is specified as a linear combination of some of the atomic orbitals enumerated at integral generation. In this module these symmetry-adapted functions will further mix to compose the solution, but as given here they must span the space of the final self-consistent orbitals. Not only must the basis orbitals of different irreducible representations be mutually orthogonal, but those of symmetry species which are components of a degenerate irreducible representation must be as well. Performed manually, this can be rather tedious. The output of `atmintc` contains this information and two support modules (`mkscf` and `mklpscf`) are also provided to ease creation of the input file in other cases.

This module allows a few, relatively primitive means of speeding or ensuring convergence of the SCF procedure. Note that a restart option is implemented so that optimization parameters can be changed with successively improved orbitals. These procedures are sometimes inadequate and, if this module fails, the `mxsscf` module can often be used to greater effect for systems of maximum total spin.

Upon convergence multipole moments (up through octapole) are automatically appended to the output file via the `dipole` module.

Table 6: Files read (above) or written (below) by `gscfnn`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`scf.inp`	User-supplied input	t
`aop.dat`	AO basis statistical info.	t
`enuc.dat`	Nuclear Energy	b
`lob.I1`	One-electron integrals	b
`lob.I2`	Two-electron integrals	b
`mos.orbs`	Molecular orbitals (opt., restart)	b
`scf.out`	Output file	t
`mos.nrgs`	MO basis statistical info.	t
`mos.orbs`	Molecular orbitals	b

Format of the `<prefix>scf.inp` File:

The input data are divided into the following sections with line numbers internal to these sections.

- General statistical and convergence parameters

  Line 1: norb nfop restart

norb	number of AOs (int)
	This must match the value from the integral generator.
nfop	number of Fock operators required for spin-coupling (int)
restart	'0' for new start
	'1' restart from previous run (requires `mos.orbs` file)

  Line 2: nsp -log(eps1) -log(eps2) -log(eps3) niter avflg shftflg prtflg dbgflg

nsp	number of symmetry species among the orbitals (int)
-log(eps1)	eps1 is the criterion for an internal eigenvalue convergence test (int)
	-log(eps1) = 13 is a common value
-log(eps2)	eps2 is the threshold for a test of orbital symmetry (int)
	-log(eps2) = 6 is often good
-log(eps3)	if the energy change falls below eps3 SCF iterations are ended (int)
	-log(eps3) = 6 is often adequate
niter	maximum number of SCF iterations (int)
avflg	'0' for the default averaging procedure (see below)
	'1' to input a custom averaging procedure
shftflg	'0' use zero shift parameter
	'1' to input a nonzero shift parameter
prtflg	'0' for no extra output
	'1' for extra printout

dbgflg	'0' for no debugging information '1' for extra debugging information

If avflg = 1, line 3: ff1 ff2 ff3 ff4 frac

ff1	'0' no average '1' average
ff2	(as ff1)
ff3	(as ff1)
ff4	(as ff1)
frac	averaging fraction (float, $0.0 \leq \text{frac} \leq 1.0$) When averaging, the sum of (frac) times the previous density and (1.0-frac) times the present converged density is used as the starting value for the density of the next iteration.

The default values (when avflag = 0) for the parameters on this line are: 0 1 1 0 0.5. This means that starting with the second iteration, averages are alternately taken then not taken for consecutive pairs of iterations. When averaging, the average of the previous density matrix and the new one is used for the next cycle. If all of the parameters ff1-ff4 are zero, no averaging is performed.

If shftflg = 1, next line: shift

shift	value added to virtual orbital energies to improve convergence (float, ≥ 0)

- Spin-coupling parameters

In this section vectors of length, nfop, are simply specified as a list of nfop values. In addition, symmetric matrices of dimension nfop x nfop, such as (with nfop = 3):

$$\begin{bmatrix} \alpha_{11} & \alpha_{21} & \alpha_{31} \\ \alpha_{21} & \alpha_{22} & \alpha_{32} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}$$

must also be specified. In the interests of efficiency, the symmetry-unique values are entered in a "lower triangular" form by listing each row in turn, only up to and including its diagonal element. This complication can be safely ignored for the closed-shell singlet parameters given as examples here because nfop is one in that case. Because the values in this section are used to calculate the number of electrons in the system, repeating fractions should be given with a large number of digits (12 is recommended).

Line 1: W(1) W(2) ... W(nfop)

W(i)	W parameter for Fock-operator i (float) For a closed-shell singlet the sole value is W(1) = 2.0.

Next nfop lines: $\boldsymbol{\alpha}$ matrix in lower triangular form

Line i of group: α(i, 1) α(i, 2) ... α(i, i)

α(i, j)	element j of row i of $\boldsymbol{\alpha}$ (float) For a closed-shell singlet the sole value is $\alpha(1,1) = 2.0$.

Next nfop lines: $\boldsymbol{\beta}$ matrix in lower triangular form

Line i of group: β(i, 1) β(i, 2) ... β(i, i)

β(i, j)	element j of row i of $\boldsymbol{\beta}$ (float) For a closed-shell singlet the sole value is $\beta(1,1) = 1.0$.

- Symmetry orbitals

nsp groups of lines: Each group gives the symmetry-adapted orbital for one symmetry species.

Line 1 of group: nfreq nocc(1) nocc(2) ... nocc(nfop) slabel

nfreq	number of orbitals in this group (int)
nocc(i)	number of group's orbitals occupied for Fock operator i (int)
slabel	symmetry species label (only used to clarify output) (string, 1-8 char.)

If nsp = 1 no symmetry is used, nfreq = norb, and no other input is required after this line. (The basis functions are constructed automatically.)
The sum of the nfreq for all the groups must equal the total number of orbitals, norb.

Next nfreq lines of group: nnzero i(1) C(1) i(2) C(2) . . . i(nnzero C(nnzero)

nnzero	number of nonzero coefficients in the orbital combination (int)
i(j)	index of the AO in nonzero term j (int)
C(j)	relative coefficient of the AO (float)

Note that only the nonzero relative values are specified and that they are automatically normalized after being read.

5.2.2 `mxsscf`

This module is an alternative R(O)HF module for systems in which the total spin is the largest allowed by the number of singly occupied orbitals (and thus can be used for closed-shell singlets). It is designed to allow more direct control over the convergence of difficult cases through use of separate energy-shift parameters for the alpha and beta orbitals. It is based on the algorithms of Guest and Saunders[15] but our implementation is relatively new. As it requires more user intervention it is often reserved as a second choice, only for systems for which `gscfnn` proves unsuitable. Table 7 contains a list of the files accessed by this module.

We have found that optimal convergence can often be obtained by restarting several times from the previous best molecular orbitals but with successively reduced shift parameters; thus an additional feature is provided. If, while the module is running, a file, `stopmxsscf` is created in the working directory (by *e.g.* the command: `touch stopmxsscf`), the program stops smoothly with the current iteration, saving the orbitals to the `<prefix>mos.orbs` file. The `stopmxsscf` file is then automatically removed before the program terminates. There is no restart flag in the input data; a restart is effected by the presence of a `<prefix>mos.orbs` file in the working directory when the job starts.

Since the input file has a format somewhat different from that of `gscfnn`, it is given a unique suffix (`scfx`). However, the `<prefix>mos.orbs` file produced is consistent with files produced by `gscfnn` and so can be used by downstream modules in a common manner. Currently, if multipole moments are desired, `gscfnn` must be restarted with the orbitals determined by `mxsscf`.

Table 7: Files read (above) or written (below) by `mxsscf`

Suffix and Extension or (Full File Name)	Description (opt.=optional)	Text (t) or Binary (b)
`scfx.inp`	User-supplied input	t
`enuc.dat`	Nuclear Energy	b
`lob.I1`	One-electron integrals	b
`lob.I2`	Two-electron integrals	b
`mos.orbs`	Molecular orbitals (opt., restart)	b
(`stopmxsscf`)	Signal to terminate gracefully	-
`scfx.out`	Output file	t
`mos.nrgs`	MO basis statistical info.	t
`mos.orbs`	Molecular orbitals	b

Format of the `<prefix>scfx.inp` File:

The input is arranged in sections similar to those of `gscfnn`. The first section has many of the same types of information, but with a different format. Since `mxsscf` is based on the number of singly and doubly occupied orbitals rather than separate Fock operators, it lacks a spin-coupling section and has a different occupation specification in the last section. The user is reminded of the helper modules already mentioned in the `gscfnn` section which simplify symmetry adapting the basis.

- General statistical and convergence parameters

  Line 1: norb

norb	number of AOs (int) This must match the value from the integral generator.

Line 2: nsp

nsp	number of symmetry species among the orbitals (int)

Line 3: niter

niter	maximum number of SCF iterations (int)

Line 4: neup

neup	maximum number of iterations allowed to produce energy rises (int)

Line 5: nbkup

nbkup	number of iterations between backup of the `mos.orbs` file (int)
	'0' for no intermediate backups

Line 6: alpha beta

alpha	initial energy shift for the alpha orbitals (float, ≥ 0)
beta	initial energy shift for the beta orbitals (float, ≥ 0)

Line 7: eps

eps	maximum energy change for SCF convergence (float)

Line 8: abfac

abfac	reduction factor for α and β if convergence occurs (float)

- Symmetry orbitals

nsp groups of lines: Each group gives the symmetry-adapted orbital for one symmetry species.

Line 1 of group: nfreq ndoub nsing

nfreq	number of orbitals in this group (int)
ndoub	number of group's orbitals doubly occupied (int)
nsing	number of group's orbitals only singly occupied (int)

Next nfreq lines of group: (same format as in `gscfnn`)

5.2.3 `uhfnn`

This module finds the spin-unrestricted, self-consistent-field solutions of the Hartree-Fock Hamiltonian using the integrals generated by either `lobeWat` or `atmintc`. (Users unfamiliar with this, perhaps, less common unrestricted-Hartree-Fock (UHF) approach to open-shell states and how it relates to the R(O)HF are referred to textbook treatments such as [16], sect. 3.8.) This module can also, optionally, perform a second-order Møller-Plesset (MP2) approximation to the correlation energy (see [16], chapt. 6). Table 8 contains a list of the files accessed by this module.

The input file for this program looks superficially like that of `gscfnn`, but there are significant differences and so it is given a different suffix (`uhf`). The format of the output orbitals is not compatible with downstream CRUNCH modules, and so, for example, it is not possible to use these orbitals in CI calculations. In fact, the concept of a conventional CI on top of a UHF calculation appears to be ambiguous.

Format of the `<prefix>uhf.inp` File:

The first input section has many of the same types of information as the corresponding `gscfnn` section, but with a different format. There is no spin-coupling section as that is foreign to the UHF method. The occupation part of the last section is somewhat different, but the necessity of symmetry adapting the basis remains. (The user is reminded of the helper modules already mentioned in connection with `gscfnn`.) The input concludes with an optional call for MP2 analysis.

- General statistical and convergence parameters

Line 1: norb restart

norb	number of AOs (int)
	This must match the value from the integral generator.
restart	'0' for new start
	'1' restart from previous run (requires `amos.orbs` and `bmos.orbs` files)

Remaining lines of section: (same as the corresponding lines for `gscfnn`)

Table 8: Files read (above) or written (below) by `uhfnn`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`uhf.inp`	User-supplied input	t
`aop.dat`	AO basis statistical info.	t
`enuc.dat`	Nuclear Energy	b
`lob.I1`	One-electron integrals	b
`lob.I2`	Two-electron integrals	b
`amos.orbs`	Alpha MOs (opt., restart)	b
`bmos.orbs`	Beta MOs (opt., restart)	b
`uhf.out`	Output file	t
`amos.orbs`	Alpha molecular orbitals	b
`bmos.orbs`	Beta molecular orbitals	b

- Symmetry orbitals

  nsp groups of lines: Each group gives the symmetry-adapted orbital for one symmetry species.

  Line 1 of group: nfreq nalpha nbeta

nfreq	number of orbitals in this group (int)
nalpha	number of group's orbitals with alpha electrons (int)
nbeta	number of group's orbitals with beta electrons (int)

  Next nfreq lines of group: (same format as in `gscfnn`)

- Request for MP2 calculation (optional)

  Line 1: mp2lab

mp2lab	'mp2' - perform MP2 calculation

5.3 Orbital Transformation

5.3.1 `matmld`

This module is an optional preparatory step for the integral transformation performed by `trannn`. There, one- and two-electron integrals are transformed from the atomic orbital (AO) basis to another orbital basis. If the desired target orbitals are simply the molecular orbitals (MOs) calculated by `gscfnn` or `mxsscf` for a single system, `matmld` is not really necessary.

Its primary use is for calculations in which the target orbitals are localized on fragments of the system (either atomic or polyatomic) and are, themselves, the solutions of separate R(O)HF calculations. In this context `matmld` constructs the matrix defining the transformation between the two bases (or equivalently the target "molecular orbitals") by fusing or melding the orbitals of several calculations. It can also form hybridized orbitals as linear combinations of the target orbitals by subsequent transformation. These are particularly useful for certain types of MCVB calculations. Optionally, the target basis can be orthogonalized. Table 9 contains the names of the files accessed by this module.

Table 9: Files read (above) or written (below) by `matmld`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`mld.inp`	User-supplied input	t
`mos.orbs`	Fragment orbitals (opt.)	b
`lob.I1`	One-electron AO integrals (opt.)	b
`mld.out`	Output file	t
`mld.orbs`	Melded, target molecular orbitals	b

When fragments make up a composite, several file-name prefixes must be distinguished. The composite prefix (which is associated with the underlying AO basis, the input and output files, and the final transformation matrix) is given on the command line when invoking the module. The fragment prefixes are given in the input file. When successive fragments are identical, transformation blocks can be reused and replicated instead of having to be reread. Thus the number of fragment files read may be less than the total number of fragments.

The atomic-orbital basis of the composite system must be the same as the combined set of fragment AO bases. The target molecular orbitals are given by the columns of the transformation matrix, while the atomic orbital basis functions are associated with its rows. (The same column-row identification applies to the fragment MO matrices of smaller dimension, as well.) The final transformation matrix must be square, that is, the sum of the number of target orbitals selected from each fragment must equal the total number of combined atomic orbitals. (The `trannn` module provides a subsequent opportunity for freezing (*i.e.* enforcing double-occupancy of) some orbitals or eliminating them completely.) The module automatically places zero coefficients between orbitals of different fragments, as appropriate. The user is warned against accidently constructing bad target orbitals. There are a few checks for some common mistakes, but no explicit detection of singular or otherwise ill-formed bases. In these circumstances downstream modules will likely fail dramatically and may not give a useful error message.

If hybrid orbitals are formed, they are automatically normalized but are not orthogonalized, unless requested. Optional Schmidt orthogonalization, accounting for AO overlap, is performed at the end and proceeds sequentially through the target molecular orbitals from first to last.

Format of the `<prefix>mld.inp` File:

The input data are divided into the following sections with line numbers internal to these sections.

- Fragment file information

  Line 1: ninp

ninp	number of fragment orbital files to be read (int)

  Next ninp lines: ffname

ffname	prefix of the `mos.orbs` file (string)

- Statistical data and flags

  Line 1: ndout num iorth hybf iprt

ndout	dimension of transformation matrix (int)
num	total number of fragments (int)
iorth	'0' for no orthogonalization
	'1' for Schmidt orthogonalization
hybf	'0' for no further transformation
	'1' for hybridization by further transformation
iprt	'0' for minimal printing
	'1' for extra output

- Source and destination of transformation coefficients

  num groups of lines: Each group describes one fragment.

  Line 1 of group: ndin ncols new

ndin	dimension of the fragment MO matrix (int)
ncols	number of columns of this matrix to be used (int)
new	'0' to reuse the previous fragment matrix
	'1' to read a new matrix from the next fragment MO file

  Line 2 of group: irp(1) irp(2) . . . irp(ndin)

irp(i)	output-matrix row corresponding to row i of the fragment matrix (int) This is the AO-to-AO mapping.

  Line 3 of group: icp1(1) icp1(2) . . . icp1(ncols)

icp1(i)	column of the fragment matrix to place in the output matrix (int) This is the source MO in the fragment.

Line 4 of group: icp2(1) icp2(2) . . . icp2(ncols)

icp2(i)	destination column in the output matrix for fragment column icp1(i) (int) This is the destination MO in the composite.

- Further orbital hybridization (optional, for hybf=1)

  If hybf = 0 this section is absent.

  The MO matrix obtained so far will be multiplied by the transformation matrix entered here. This matrix is just the identity except for the columns specified here and these columns are zero except for the terms explicitly given.

  Line 1: nlc

nlc	number of hybrids to form (int)

Next nlc lines: ncol nnzr row(1) a(1,ncols) row(2) a(2,ncols) . . . row(nnzr) a(nnzr,ncols)

ncol	column number of the composite MO which will be the hybrid (int)
nnzr	number of nonzero coefficients in the transformation-matrix column (int)
row(i)	row of nonzero coefficient i of the transformation matrix (int)
a(i,ncol)	relative nonzero coefficient i of the transformation matrix (float)

5.3.2 `trannn`

This module begins with the one- and two-electron integrals produced in an atomic-orbital (AO) basis by `lobeWat` or `atmintc` and transforms them to an arbitrary basis of orbitals. (For a brief discussion in the context of a simple example, see Ref. [16], p. 164.) Table 10 contains the names of the files accessed by this module.

Table 10: Files read (above) or written (below) by `trannn`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`trn.inp`	User-supplied input	t
`prm.inp`	Orbital permutation (opt.)	t
`mld.orbs/mos.orbs`	Molecular orbitals (opt.)	b
`enuc.dat`	Nuclear energy	b
`lob.I1`	Original one-electron integrals	b
`lob.I2`	Original two-electron integrals	b
`lob.I2n`	Original two-electron integrals info.	t
`trn.out`	Output file	t
`trn.dns`	Transformed density matrix	b
`trn.orbs`	Transformed molecular orbitals	b
`core.nrg`	Nuclear energy (after transformation)	b
`trn.I1`	Transformed one-electron integrals	b
`trn.I2`	Transformed two-electron integrals	b
`trn.I2n`	Transformed two-electron integrals info.	t

For brevity, and in accord with the way `trannn` is commonly employed, this basis is referred to in this section as the *molecular-orbital (MO) basis*, but it should be emphasized that it is not required to correspond to Hartree-Fock solutions of the system or any portion of it. The transformation can be defined by a matrix of coefficients of the new MOs in terms of the original atomic orbitals. For clarity, in the input specification below, the MOs will be associated with the columns of this and related matrices and the AOs with the rows. The new basis functions can be selected from the columns of the MO matrix contained in the binary orbital files produced by `gscfnn`, `mxsscf`, or `matmld` and/or specified explicitly. It is the user's responsibility to insure that the new basis is not singular or otherwise malformed. No explicit checks are performed by this or following modules, but catastrophic and often puzzling results are likely to occur.

Unless all of the MOs are specified explicitly in the input file, `trannn` seeks first a binary molecular orbital file produced by `matmld`. If it finds none, it then looks for one from the Hartree-Fock modules, `gscfnn` or `mxsscf`. The user is cautioned to carefully clean working directories of old intermediate files to avoid any confusion.

This module can also produce what are referred to as *core orbitals*. These are fixed and always doubly occupied hereafter. They are effectively removed from the active orbital space (with their energies automatically incorporated into the transformed nuclear-repulsion energy). The user may specify these arbitrarily, with no necessary correspondence to conventional, filled electronic shells.

For later use by the `symgenn` module, there is also provision for global orbital permutation. This is specified in an additional input file, `<prefix>prm.inp`. If the file is absent, there is no permutation. If permutation is to be performed, this file must be present and unchanged when both `trannn` and `symgenn` are executed. The file contains space-separated integers: i(1) i(2) ... i(norb) where i(j) is the number of the orbital to be moved to position j in the new ordering. Core orbitals are not included in the numbering . (That is, the first orbital outside the core is in position 1 and there should be norbo−ncore numbers in the `<prefix>prm.inp` file ranging from 1 to norbo−ncore.)

Format of the `<prefix>trn.inp` File:

The input is organized into groups with line numbers internal to the groups.

- General statistical parameters

  Line 1: norbi norbo ncore nstr iin nbold iprnt

norbi	number of AOs in the basis of the input integrals (int)
norbo	number of MOs output, including core orbitals (int, ≤ norbi)
ncore	number of the norbo MOs to be placed in the core (int, ≤ norbo) The core orbitals appear first in the sections below.
nstr	this is a parameter no longer used, set it to ‘0’
iin	‘0’ if all of the MOs are given explicitly in the input ‘1’ if at least some MOs come from a binary orbital file
nbold	this is a parameter no longer used, set it to ‘0’
iprnt	‘0’ for normal printout ‘1’ for extra printout

  Note that for the input to downstream modules, the orbital basis after `trannn` contains norbo−ncore orbitals.

- Selection of orbitals from previous calculation (optional for iin=1)

  If iin=0 this section is absent.

  Line 1: nmocc

nmocc	number of columns to be taken from the binary orbital file (int)

  Line 2: c(1) c(2) ... c(nmocc)

c(i)	column of the orbital-matrix to use as output MO number i (int)

- User-specified orbitals (optional for norbo > nmocc)

  If nmocc = norbo all the orbitals have been specified and this section is absent.

  norbo − nmocc lines: Each line defines an MO.

  There are two alternative forms for specifying the MOs: either all the coefficients can be given or just the positions and values of the nonzero elements. Each MO can be given in either format which is automatically detected.

  Form 1: norbi a(1) a(2) ... a(norbi)

norbi	number of orbitals in the AO basis (int, same as above)
a(i)	coefficient of AO i in the MO (float)

  Form 2: nnz r(1) a(1) r(2) a(2) ... r(nnz) a(nnz)

nnz	number of AOs with nonzero contributions to the MO (int)
r(i)	position in the AO list of nonzero contribution i (int)
a(i)	coefficient of AO number r(i) in the MO (float)

5.3.3 `ctran`

This special-purpose module begins with the integrals produced by `atmintc`, `lobeWat`, or `trannn` and performs the transformation necessary to use the $\mathrm{D_{\infty h}}$ or $\mathrm{C_{\infty v}}$ point groups in CI calculations of linear molecules or atoms. Table 11 contains the names of the files accessed by this module.

Table 11: Files read (above) or written (below) by `ctran`

Suffix and Extension	Description	Text (t) or Binary (b)
`ctrn.inp`	User-supplied input	t
`core.nrg/enuc.dat`	Nuclear energy	b
`trn.I1/lob.I1`	One-electron integrals	b
`trn.I2/lob.I2`	Two-electron integrals	b
`trn.I2n/lob.I2n`	Two-electron integrals info.	t
`ctrn.out`	Output file	t
`ctrn.I1`	Transformed one-electron integrals	b
`ctrn.I2`	Transformed two-electron integrals	b
`ctrn.I2n`	Transformed two-electron integrals info.	t

The transformation begins from pairs of real orbitals of the form:

$$u(z,\rho,\phi) = R(z,\rho)\,T(\phi)$$

where:

$$T(\phi) = \cos(\mathrm{m}\phi) \text{ and } \sin(\mathrm{m}\phi)$$

and z is the coordinate along the angular-momentum quantization axis, ρ is the radial coordinate, and ϕ is the angle around the axis. A nonnegative integer, m must be the same for both starting components. These orbitals belong to the real symmetry species (refer back to section 4.3) of the irreducible representations of the continuous linear groups. This module converts the one- and two-electron integrals in the original basis of pairs of these functions into those which would arise from the corresponding, normalized complex orbitals:

$$u'(z,\rho,\phi) = \frac{1}{\sqrt{2}}\,S_{\mathrm{m}'_{\mathrm{j}}}\,R(z,\rho)\,e^{i\mathrm{m}'_{\mathrm{j}}\phi},$$

which are now eigenfunctions of the angular momentum projection operator $\hat{\mathrm{L}}_z$ with corresponding quantum numbers $\mathrm{m}'_1 = \mathrm{m}$ and $\mathrm{m}'_2 = -\mathrm{m}$. An overall sign change $S_{\mathrm{m}'_{\mathrm{j}}}$ may also be introduced. As implemented here, the basic formula is:

$$R(z,\rho)\{S_c\,cos(\mathrm{m}\phi) + i\,S_s\,sin(\mathrm{m}\phi)\} = R(z,\rho)\,S_{\mathrm{m}'_{\mathrm{j}}}\,e^{i\mathrm{m}'_{\mathrm{j}}\phi}$$

where S_c and S_s are ± 1 and, along with m'_{j}, are specified in the input, while $S_{\mathrm{m}'_{\mathrm{j}}}$ is a consequence of the transformation. Care must be taken in its use, as the module lacks any internal check to confirm the user's assertion that the two real functions to be combined share the same $R(z,\rho)$ and m.

This module cannot be used to prune the number of orbitals or perform more general basis transformations; these tasks must be accomplished by previous application of `trannn`.

The program first seeks integral files produced by `trannn`. If these are absent, it then tries to find those from `atmintc` or `lobeWat`. Careful directory cleaning is again recommended.

Once integrals over complex functions have been formed, the downstream configuration-selection and CI-matrix-construction modules `symgenn`, `omtrcIII`, and `nomtrcII` must be invoked with special flags to indicate the complex basis and linear point group. Note that only some of the file-name suffixes and extensions in this and later modules are unique for the complex basis. Unpredictable results (likely without meaningful errors) will occur if the real/complex bases and discrete/linear versions of the modules are mixed erroneously. Careful cleaning is again recommended and complete calculations in real and complex bases should probably be performed in separate directories or, at least, use different prefixes. MP2 and MCSCF calculations in a complex basis are not currently supported.

Format of the `<prefix>ctrn.inp` File:

Line 1: norb

norb	number of orbitals (excluding the `trannn` core) (int)

Next norb lines: mp cost sint(optional)

mp	the $\hat{L}_z$ quantum number, m$'$, for the complex orbital (int)
cost	the cosine term in the complex orbital (int) Two data are encoded: sign(cost) gives S_c and abs(cost) is the orbital number with cosine symmetry.
sint	the sine term in the complex orbital (int) Same format as cost, but missing if mp=0.

5.4 Configuration Selection and Symmetry Projection

5.4.1 `symgenn`

This module is the heart of the multi-configurational methods in CRUNCH. Its basic function is to select configurations of a particular spin and symmetry-adapt them to the point group irreducible representation (IR) specified by the user. Flexible selection criteria can be applied which are capable of producing large numbers of configurations with a single set of rules or of narrowly targeting multi-electron basis functions, even to the level of individual configurations. Table 12 contains the names of the files accessed by this module.

Table 12: Files read (above) or written (below) by `symgenn`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`sym.inp`	User-supplied input	t
`prm.inp`	Orbital permutation (opt.)	t
`sym.out`	Output file	t
`sym.lu1`	CI configurations	b
`sym.lu8`	CI statistical info.	t

The selection criteria can be divided into two broad classes based on their scope. The spin and spatial symmetry are specified globally and apply to all the functions produced by the module. (Thus, calculating another spin state or symmetry requires separate application of the module.) Currently the maximum spin supported is S=3 (septets). The other criteria are given in any number of rule sets which are applied successively, each producing a sublist of allowed configurations. All rules in a set must be satisfied in order for a configuration to be added to the sublist. These variable rule sets require further discussion.

A *charge center* is a subset of the orbitals (possibly including core orbitals) and the nuclei. Each orbital and nucleus is assigned to one and only one charge center globally. The occupancy of the active orbitals can, however, be variously limited by specifying possibly different minimum and maximum total charges for the charge centers in each section of rules.

Configurations can also be constrained by defining an initial number of the orbitals as *base orbitals* and setting the number of electrons that will be excited out of this set. The total number of doubly occupied orbitals can also be limited by imposing maximum and minimum values. User-specified groups of orbitals required to hold a definite number of electrons can be defined and the occupancy of individual orbitals can even be arbitrarily freed or limited.

The user is reminded of the requirement that the orbitals composing the configurations provide a basis for a real generalized permutation representation (see Section 4.1). This means that under all the operations of the point group, an orbital, whether real or complex, must be transformed into one of the other orbitals, or into the negative of an orbital. (This may limit the point group of the calculation to a subgroup of the nuclear-framework point group.)

This requirement allows all the information contained in the representation matrices, $\mathbf{D}(\hat{R})$, to be conveniently expressed in the form of an orbital transformation table (*c.f.* Table 3 of section 4.2). If the symmetry elements, beginning with the identity, are assigned to the columns of the table, and the orbitals to the rows, then the effect of each operator on each orbital can be simply expressed as an entry which is the index of the produced orbital, with a prepended minus sign if it is inverted. These are the parameters: te t(2) t(3) . . . t(nspop) below.

It may seem odd to the user that, aside from the initial position reserved for the identity, the ordering of symmetry elements in the input is arbitrary. In fact the module makes no use of the actual nature of the operation, but only of the existence of an operator with the indicated effect. It may help the user avoid confusion, however, to hew to the order of elements in some standard reference. Constructing the orbital transformation table is one of the most tedious and error-prone steps of forming the input file. The `synp` helper module is especially commended to the user.

See the `trannn` section for the format of an optional `<prefix>prm.inp` file which has the effect of reordering the orbitals given in the input files. It is desirable to have a means of doing this rapidly (and repeatedly), without having to manually change the `<prefix>trn.inp` and `<prefix>sym.inp` files, as the actual multi-configurational basis produced by `symgenn` can depend upon the order in which the orbitals are listed (see [3] and [17] for more details).

The extreme flexibility of this module is accompanied with some complexity and potential for error. As the succeeding configuration-interaction steps can involve considerable computational expense, this module is designed to be separate and fast, so that debugging can be performed before proceeding. (Note that it does not even read files produced by modules usually run before it.) The user should employ extra care here and consider performing preliminary CI calculations with a limited, representative set of configurations (perhaps by limiting the degree of excitation) before committing to the ultimate calculation of interest. As a further aid the module `symgerr` is automatically called at the end and checks the configuration list for certain classes of errors.

Finally, a brief word about the output. We have been a bit cavalier with the term "configuration" heretofore. The output is given in terms of tableaux, symmetry functions, and constellations. Without repeating the more complete discussion of reference [3] (see especially chapter 6), the (Young's) tableaux are the anti-symmetrized association of electrons with spin-orbitals. The symmetry functions are the symmetry-adapted combinations of tableaux which form the multi-electron basis functions of the configuration-interaction problem. In the variety of spin-free quantum chemistry used here they can be regarded as the "high-spin" $\mathrm{m_s} = \mathrm{S}$ basis functions. Finally, constellations are sets of tableaux interrelated by the spatial symmetry operators.

In light of the module's complexity, the discussion immediately following is limited to discrete symmetry point groups. Rather than burden a unified treatment with multiple excursions and exceptions, the changes to the input required for the continuous groups $\mathrm{C_{\infty v}}$ and $\mathrm{D_{\infty h}}$ are segregated to a separate section at the end.

Format of the `<prefix>sym.inp` File for Discrete Point Groups:

For discrete point groups invoke the code with the command `crunch -s`.

The input data are divided into the following sections with line numbers internal to these sections.

- Title

  Line 1: title

title	description of the problem (string, 1-80 char.) This must be the first *noncomment* line and may contain spaces.

- General statistical parameters

  Line 1:

nele	number of electrons outside the `trannn` core (int)
mult	the spin multiplicity, 2S+1 (int, ≤ 7)
norb	number of orbitals outside the `trannn` core (int)
nspop	number of operations in the spatial group (int)
nchgcnt	number of charge centers (int)
intgr	this is a parameter no longer used, set it to '0'
style	'0' for tableau output with alphanumeric orbital labels '1' for more compact output for larger problems

- Orbital labels and symmetry

  norb lines: orbv te t(2) t(3) . . . t(nspop)

orbv	orbital label (for tableau printout) (string, 1-7 char.)
te	position of orbital in list (int)
	(also, of course, the orbital produced by the identity operator)
t(j)	number of the orbital produced by symmetry operator j (int)
	This is negative if the product orbital is inverted.

- Symmetry of the multi-electron configurations

  Line 1: fd(1) fd(2) . . . fd(nspop)

fd(j)	character of the symmetry species under symmetry operator j (float)
	The order of the operators must be the same as in the previous section.

  Refer back to section 4.3 for the definition of a symmetry species.
  When the irreducible representation (IR) is degenerate, the character of the entire IR can be given (fd(1) is then equal to the IR degeneracy) in which case functions of all the degenerate symmetry species will be spanned by the configurations. Alternatively, functions of only one of the degenerate components can be requested (in this case, fd(1) will be 1.0).

- Charge center definition

  Line 1: chg(1) chg(2) . . . chg(nchgcnt)

chg(i)	charge of charge center i before active-orbital occupation (int)
	(Each doubly occupied core orbital of the center reduces
	the nuclear charges by two).

  Line 2: cnt(1) cnt(2) . . . cnt(norb)

cnt(i)	charge center that active orbital i belongs to (int)

- Variable Configuration Rule Sets

  Any number of six-line groups give rules which constrain the orbital occupations of each associated sublist.

  Line 1 of group: nbase nexc pmin pmax label

nbase	number of initial orbitals from which excitations may be made (int)
	If '0' the module is terminated and no more data is read.
nexc	number of electrons excited out of the base orbitals (int)
pmin	minimum number of doubly occupied orbitals (int)
pmax	maximum number of doubly occupied orbitals (int)
label	label to identify rule set and sublist (string, 1-80 char.)
	As this is white-space-terminated, '_' can be used to connect words.

  Line 2 of group: qmin(1) qmin(2) . . . qmin(nchgcnt)

qmin(i)	minimum combined charge on charge-center i (int)
	(*i.e.* the minimum allowed for chg(i) minus the number
	of active electrons occupying the charge center)

  Line 3 of group: qmax(1) qmax(2) . . . qmax(nchgcnt)

qmax(i)	maximum combined charge on charge-center i (int)

  Line 4 of group: set(1) set(2) . . . set(norb)

set(i)	the orbital set that orbital i belongs to (int)
	The module automatically counts the number of sets assigned;
	do not skip any numbers in enumerating the sets.
	(If all are equal to '1,' there is just one set.)

  Line 5 of group: mset(1) mset(2) . . . mset(nset)

mset(i)	the number of electrons occupying orbitals in set i (int)
	nset is the largest of the set(i).
	The sum of all the mset must equal nele.

  Line 6 of group: occflg(1) occflg(2) . . . occflg(norb)

occflg(i)	the occupation restriction for orbital i (int, $1 \leq$ occflg(i) ≤ 7)

These occupation restriction indexes are essentially a binary encoding of the possible orbital occupancies. Allowed double occupation contributes 4 (2^2 = 100(binary)) to the index, single occupation 2 (2^1 = 010(binary)), and no occupation 1 (2^0 = 001(binary)). Multiple possibilities are indicated by summing the contributions. Thus, '3' (011(binary)) would indicate that no or a single electron (but not two electrons) could occupy the orbital. Definite double occupancy would be indicated by '4' while no occupation would be indicated by '1.' If the orbital occupation is not restricted, use '7' (111(binary))

Changes to the Format of `<prefix>sym.inp` for $C_{\infty v}$ and $D_{\infty h}$ Point Groups:

To use these point groups the code must be invoked with: `crunch -S`. (Mistakenly using the parameter for the discrete groups will produce unpredictable errors.)

This module does not employ a direct or complete treatment of the continuous point groups but brings them into effect by combining specification of the angular-momentum-projection quantum number (usually denoted m_l) with a subset of the symmetry operations sufficient to distinguish the irreducible representation. This allows use of the `symgenn` algorithms with only modest modification. Note that, as $D_{\infty h}$ is a proper subgroup of O(3), this module can be used to resolve much, but not all, of the symmetry in atomic calculations.

Only the following sections need to have their formats altered from what has just been described. (In particular, the variable rule sets are unchanged.) Note that unlike the discrete case, a definite order is required for the symmetry operations.

- General statistical parameters

  For $C_{\infty v}$ nspop is 2 and for $D_{\infty h}$ it is 4.

- Orbital labels and symmetry

  If nspop = 2 ($C_{\infty v}$):

  norb lines: orbv ml te t(σ_{xz})

orbv	orbital label (for tableau printout) (string, 1-7 char.)
ml	angular-momentum-projection quantum number of orbital (int)
te	position of orbital in list (int)
	(also, of course, the orbital produced by the identity operator)
t(σ_{xz})	number of the orbital produced by the xz reflection (int)
	This is negative if the product orbital is inverted.

  Or if nspop = 4 ($D_{\infty h}$):

  norb lines: orbv ml te t(σ_{xz}) t(i) t(C_{2y})

orbv	orbital label (for tableau printout) (string, 1-7 characters)
ml	angular-momentum-projection quantum number of orbital (int)
te	position of orbital in list (int)
	(also, of course, the orbital produced by the identity operator)
t(σ_{xz})	number of the orbital produced by the xz reflection (int)
	This is negative if the product orbital is inverted.
t(i)	number of the orbital produced by the inversion operator (int)
	This is negative if the product orbital is inverted.
t(C_{2y})	number of the orbital produced by C_2 rotation about the y-axis (int)
	This is negative if the product orbital is inverted.

- Symmetry of the multi-electron configurations

  Line 1: Ml fd(1) fd(2) ... fd(nspop)

Ml	angular-momentum-projection quantum number for configurations (int)
fd(j)	character of the symmetry species under symmetry operator j (float)
	The order of the operators is as given in the previous section.

  The magnitude of the characters is a bit different in this case. Functions of only one member of a degenerate IR may be produced. Although the characters of the entire IR are given (fd(1) is then '2'), the sign of the quantum number determines which of the two components is obtained.

5.5 Configuration-Interaction with Orthogonal Orbitals

5.5.1 `mp2`

This module calculates the second-order, Møller-Plesset (MP2) contribution to the correlation energy for restricted, closed- and open-shell systems (for an overview, see [16], chap. 6). It follows transformation (by `trannn`) of the integrals over atomic orbitals to the real R(O)HF orbitals calculated by either `gscfnn` or `mxsscf`. Table 13 contains the names of the files accessed by this module.

Table 13: Files read (above) or written (below) by `mp2`

Suffix and Extension	Description (opt.=optional)	Text (t) or Binary (b)
`mp2.inp`	User-supplied input	t
`core.nrg`	Nuclear Energy	b
`trn.I1`	One-electron integrals	b
`trn.I2`	Two-electron integrals	b
`trn.orbs`	Molecular orbitals (start or restart)	b
`mp2.out`	Output file	t

This module is in the class of *configuration state function* approaches to Møller-Plesset perturbation theory. As exemplified by Murray and Davidson[18] these use pure spin states for the excited configuration functions. We implement these as tableau functions. Murray and Davidson outline two variants, labeled OPT1 and OPT2, which differ in the definition of the unperturbed Hamiltonian; both are supported here.

The MP2 procedure for closed-shell systems is unambiguous. For open-shell systems complications arise from differences in orbital canonicalization at the Hartree-Fock step (see [19] for a short review). Guest and Saunders[15] describe one popular choice which we have used in `mxsscf`. The `gscfnn` module is much more flexible, but the simplest W, α, or β parameters give Roothaan[13] canonicalization.

Canonicalization differences lead not only to different MP2 energies but also to different orbital energies, and sometimes even different energy ordering among the orbitals. This module requires that `trannn` has previously been used to order the orbitals as doubly occupied, followed by singly occupied, followed by virtual, regardless of their energy ordering. In particular, for triplet and higher spin states, Roothaan canonicalization may not produce this ordering automatically at the Hartree-Fock step. The user must verify or produce the required ordering, as the module makes no independent check.

For open-shell cases the output separates the contributions into different excitation patterns and also lists what we believe are the corresponding designations of the ZAPT procedure used in GAMESS[6].

Format of the `<prefix>mp2.inp` File:

Line 1: iprt

iprt	'0' gives minimal output
	'1' gives extra output

Line 2: iopt

iopt	'1' uses OPT1
	'2' uses OPT2
	This parameter is ignored for closed-shell systems (isomo = 0 below).

Line 3: idomo

idomo	number of doubly occupied orbitals outside the `trannn` core (int)

Line 4: isomo

isomo	number of singly occupied orbitals (int)

5.5.2 `omtrcIII`

This module constructs (but does not diagonalize) the CI matrix for the configurations produced by `symgenn`. It relies on the configuration orbitals being orthogonal. Table 14 contains the names of the files accessed by this module.

Table 14: Files read (above) or written (below) by `omtrcIII`

Suffix and Extension	Description	Text (t) or Binary (b)
`sym.lu1`	CI configurations	b
`sym.lu8`	CI statistical info.	t
`core.nrg`	Nuclear Energy	b
`trn.I1/ctrn.I1`	One-electron integrals	b
`trn.I2/ctrn.I2`	Two-electron integrals	b
`trn.I2n/ctrn.I2n`	Two-electron integrals info.	t
`mat.out`	Output file	t
`mat.III`	Hamiltonian matrix in orthogonal basis	b
`trn.III`	`symgenn`/orthogonal basis transformation	b
`obs.III`	Overlap matrix in `symgenn` basis	b

When the orbitals are real, the discrete-point-group version of `symgenn` should have been used to create the configurations and `omtrcIII` should be invoked with `crunch -o`. When `ctran` has been used to produce complex orbitals, `crunch -SO` invokes the correct versions of `symgenn` and `omtrcIII`. There are no internal checks that the right versions are called; that is the user's responsibility. The code also assumes (without checking) that the orbitals are indeed orthogonal. Bad results, possibly without a meaningful error message, will occur if this is not the case.

This module does not require a user-supplied input file.

5.5.3 `mcscf`

This module performs a multiconfigurational self-consistent-field (MCSCF) calculation in which the molecular orbitals (MOs) and the CI coefficients of configurations containing these orbitals are simultaneously optimized (see [16], section 4.5 for a brief description). It is based upon the algorithms of Werner and Meyer[20], although it is not a complete implementation and especially lacks some of their methods for dealing with difficult convergence. Table 15 contains the names of the files accessed by this module. (Although this might look like a great number, the user can be reassured that several (in particular those with `<prefix>syma.lu1`, `<prefix>emm.int`, `<prefix>den.G1`, and `<prefix>den.G2`) are intermediate files written and read by the module, or modules automatically called by it, in a manner invisible to the user.)

The input data can be somewhat involved and small changes can have large effects on the progress of the calculation. In addition to the input described below, the electron configurations must be determined in a separate, preceeding run of `symgenn` using the same file-name prefix.

When it converges, the multidimensional Newton-Raphson method (see [21], chap. 9) used here for root finding manifests second-order convergence. However, difficult cases do arise when it diverges. This problem is compounded by the fact that R(O)HF orbitals often provide poor initial guesses for the MCSCF orbitals and provision has therefore been made for a variety of starting orbitals.

In its most direct usage, the initial molecular orbitals are the R(O)HF orbitals taken from the binary orbital file produced by `gscfnn` or `mxsscf`. It is also possible to start a calculation from the orbitals given in the `<prefix>mchf.orbs` file from a previous `mcscf` run. This enables restart in the case of failed convergence and even manipulation of the underlying orbital space between runs in an effort to encourage convergence. In this latter case orthonormality of the MOs is enforced before optimization begins. Finally, symmetrically orthogonalized functions of the underlying basis (atomic orbitals (AOs), unless `trannn` has been used in advance) may be used as the starting molecular orbitals. This is provided in case the R(O)HF orbitals cause optimization to diverge and in ideal cases will produce convergence in fewer than ten iterations. See the parameter infile below to choose between these three cases.

The module can be forced to gracefully exit at the end of the current iteration by creating a `stpmcscf` file in the working directory (*i.e.* with the command `touch stpmcscf`). When detected the module writes the current MOs to the `<prefix>mchf.orbs` file, deletes `stpmcscf` and exits.

As a matter of convenience, the molecular orbitals are divided into a *foundation* space, an *active* space,

Table 15: Files read (above) or written (below) by `mcscf`

Suffix and Extension or (Full File Name)	Description	Text (t) or Binary (b)
`mchf.inp`	User-supplied input	t
`sym.lu1`	CI configurations	b
`sym.lu8`	CI statistical info.	t
`core.nrg/enuc.dat`	Nuclear Energy	b
`trn.I1/lob.I1`	One-electron integrals	b
`trn.I2/lob.I2`	Two-electron integrals	b
`mos.orbs/mos.mchf`	Molecular orbitals (start or restart)	b
(`stpmcscf`)	Signal to terminate gracefully	-
`syma.lu1`	Modified CI statistical info.	b
`emm.int`	Electrostatic moments	t
`den.G1`	First-order density matrix	t
`den.G2`	Second-order density matrix	t
`mchf.out`	Output file	t
`mos.mchf`	Molecular orbitals	b
`mchf.gam`	Gamma matrix	t
`syma.lu1`	Modified CI statistical info.	b
`emm.int`	Electrostatic moments	t
`den.G1`	First-order density matrix	t
`den.G2`	Second-order density matrix	t

and an *empty* space. The optional foundation orbitals are assumed to be doubly occupied in all configurations and are not even mentioned in the `symgenn` input, but are assumed to "lie beneath" the orbitals specified there. The active orbitals are those manipulated and possibly occupied in the `symgenn` configurations. The orbitals of both the foundation and active spaces are subjected to optimization, but the foundation orbitals are always doubly occupied. The empty orbitals, if any, are assumed to be unoccupied in all configurations (and are also not mentioned in the `symgenn` input) but complete the basis space in which the orbital optimization is performed.

In order to freeze doubly occupied R(O)HF molecular orbitals with respect to optimization, `trannn` must have been used previously to place them in the core and thus eliminate the corresponding functions from the basis of the integrals. At present this requires that the symmetrically orthogonalized functions be used for the initial guess (infile = '2' below).

Integrals from `trannn` are sought first. If these are absent, `lobeWat` or `atmintc` integrals are expected. Careful cleaning will prevent confusion. (At present, calculations using the complex orbitals from `ctran` are not supported.)

The specification of symmetry in this module might seem a bit odd. For the orbitals the transformation properties of the symmetry species do not need to be specified, but only which orbitals transform in the same way. If there are nsym symmetry species, they must be given distinct and successive (but otherwise arbitrary) indices between 1 and nsym with orbitals assigned the appropriate index. (Additionally the active-orbital symmetry indices given here must not contradict those implied by the orbital transformation table in the `symgenn` input.) The symmetry of the MCSCF state is specified as the symmetry of the basis configurations in the `symgenn` input in the usual way. (If necessary, revisit the definition of a symmetry species in section 4.3; recall that different components of a degenerate irreducible representation are regarded as different symmetry species.)

At the end of the calculation multipole electrostatic moments (up through octapoles) for the lowest energy state are calculated.

Format of the `<prefix>mchf.inp` File:

The input data are divided into the sections given below. Line numbers are internal to these sections. Remember that the electron configurations are assumed to have been determined by a separate, preceeding `symgenn` calculation using the same file-name prefix.

- General statistical and convergence information

  Line 1: nfound nactiv nstrto iprt

nfound	number of foundation molecular orbitals (int)
nactiv	number of active MOs (int)
nstrto	this is a parameter no longer used, set it to '0'
iprt	'0' - no extra output '1' - voluminous output

  Line 2: norb infile

norb	number of basis orbitals (number of AOs minus `trannn` core orbitals) (int)
infile	'0' - initial MOs come from an `mos.orbs` file '1' - initial MOs come from an `mchf.orbs` file '2' - initial MOs are orthogonalized basis orbitals

  Line 3: uuiter ncycles cutoff

uuiter	this parameter is not used, set it to '0'
ncycles	maximum number of iterations allowed (int)
cutoff	convergence threshold for root-mean-squared energy gradient (float) 1.0e-6 works in many cases.

- Orbital Specification

  Line 1: iorb(1) iorb(2) ... iorb(norb)

iorb(i)	the number of the MO in `mos.orbs` which is the initial guess for MCSCF orbital number i (int) This old-to-new MO mapping is always read, but is ignored unless infile = '0'. The first nfound orbitals are foundation, the next nactiv are active, and the remainder are empty.

  Line 2: sorb(1) sorb(2) ... sorb(norb)

sorb(i)	symmetry species index for MCSCF orbital number i (int)

  Line 3: dorb(1) dorb(2) ... dorb(norb)

dorb(i)	'0' - MCSCF orbital i is doubly occupied in all configurations '1' - MCSCF orbital i is not always doubly occupied These values relate to `symgenn` configurations, not whether an orbital is in the foundation.

  Line 4: lorb(1) lorb(2) ... lorb(nsym)

lorb(i)	label for symmetry species of index i (string, 1-8 char.) These are only used to help clarify the output. nsym is the largest of the sorb(i) above.

5.6 Nonorthogonal-Orbital CI Matrix Construction

5.6.1 `nomtrcII`

This module constructs (but does not diagonalize) the CI matrix for the configurations constructed by `symgenn`. It makes no assumptions about the orthogonality of the orbitals. The methods used are described in [3], especially chapters 5 and 6. Table 16 contains the names of the files accessed by this module.

The algorithms used in `omtrcIII` are faster then those employed here because of the ability to rely on the orbitals being orthogonal. Nothing prevents this module from being applied to an orthogonal case, but that should ordinarily be avoided (unless, for example, an independent confirmation of `omtrcIII` results is desired).

When the orbitals are real, the discrete-point-group version of `symgenn` should have been used to create the configurations and `nomtrcII` should be invoked with `crunch -m`. When `ctran` has been used to produce complex orbitals, `crunch -SM` invokes the correct versions of `symgenn` and `nomtrcII`. There are no internal checks that the right versions are called; that is the user's responsibility. Bad results, possibly without a meaningful error message, can occur otherwise. For the real case integral files produced by `trannn` are sought first. If these are absent, those from `atmintc` or `lobeWat` are used. Careful directory cleaning is again recommended.

This module does not require a user supplied input file.

Table 16: Files read (above) or written (below) by `nomtrcII`

Suffix and Extension	Description	Text (t) or Binary (b)
`sym.lu1`	CI configurations	b
`sym.lu8`	CI statistical info.	t
`enuc.dat/core.nrg`	Nuclear Energy	b
`trn.I1/lob.I1/ctrn.I1`	One-electron integrals	b
`trn.I2/lob.I2/ctrn.I2`	Two-electron integrals	b
`trn.I2n/lob.I2n/ctrn.I2n`	Two-electron integrals info.	t
`mat.out`	Output file	t
`mat.HII`	Hamiltonian matrix	b
`mat.SII`	Overlap matrix	b
`nrm.SII`	Overlap matrix diagonal elements	b
`core.nrg`	Nuclear Energy	b

5.7 CI Eigensolution

5.7.1 `neweig`

This module finds the eigenvalues and eigenvectors of the CI matrices produced by `omtrcIII` and `nomtrcII` but is limited to small- to medium-size problems. Table 17 contains the names of the files which may be accessed by this module, with the exception of a few intermediate working files which are deleted upon successful completion.

Table 17: Files read (above) or written (below) by `neweig`

Suffix and Extension	Description	Text (t) or Binary (b)
`mat.HII`	Hamiltonian matrix (nonorth.)	b
`mat.SII`	Overlap matrix (nonorth.)	b
`mat.III`	Hamiltonian matrix (orth.)	b
`trn.III`	Symgenn basis transformation (orth.)	b
`sym.lu8`	CI statistical info.	t
`core.nrg`	Nuclear Energy	b
`eig.out`	Output file	t
`eig.hs`	Normalized H and S matrices	b
`eig.nrg`	Eigenvalues	b
`eig.vec`	Eigenvectors	b

When this module diagonalizes a CI matrix produced by `nomtrcII` (the *nonorthogonal* case) it solves the generalized-eigenvalue problem (see [21], section 11.0 for a brief description):

$$\mathbf{H}\overrightarrow{x_i} = E_i\mathbf{S}\overrightarrow{x_i}$$

where the overlap matrix, $\mathbf{S}$, is not necessarily equal to the identity. When the orbitals which make up the configurations are orthogonal and `omtrcIII` has been used to generate the matrix (the *orthogonal* case), there is some economy as the overlap matrix can be assumed to be the identity.

The binary files which contain the matrices (and related information) produced by `omtrcIII` and `nomtrcII` have unique names. This module seeks first the products of `nomtrcII` and, if that fails, those of `omtrcIII`. Be sure to clean carefully so that the files used are those intended.

This module is capable of finding any number of the lowest energy eigenvalues and eigenvectors of the CI matrix, but is not particularly fast, especially as the matrix size grows. The alternative `beigII` module is more suitable for larger problems for which only a handful of the lowest roots are desired.

As of the beginning of 2011, a standard desktop machine with 2 GB of memory can, in principle, use this module to find the roots of a CI matrix involving around eight-thousand symmetry functions (although that may take a while). Problems with around a thousand configurations can be solved fairly quickly.

This module does not require a user-supplied input file.

5.7.2 `beigII`

This module finds a few of the lowest eigenvalues and eigenvectors of CI matrices produced by `omtrcIII` and `nomtrcII` and is to be preferred over `neweig` for large problems.

We continue in the context of our discussion of the generalized-eigenvalue problem begun in the `neweig` section. For the nonorthogonal case (when `nomtrcII` has produced the matrix), a separate, initial module, `sinv`, is automatically called which diagonalizes the overlap matrix. Table 18 contains the names of the files which may be read or written by `sinv`, with the exception of a few intermediate working files which are deleted upon successful completion. Then the main module, `beigII` begins. The orthogonal case involves `beigII` only. Table 19 contains the names of the files which may be accessed by this module.

Table 18: Files read (above) or written (below) by `sinv`

Suffix and Extension	Description	Text (t) or Binary (b)
`sym.lu8`	CI statistical info.	t
`mat.SII`	Overlap matrix	b
`sm1.mat`	Overlap inverse	b
`mat.siok`	Signal for successful S inversion	-

Table 19: Files read (above) or written (below) by the main portion of `beigII`

Suffix and Extension	Description	Text (t) or Binary (b)
`big.inp`	User-supplied input	t
`core.nrg`	Nuclear Energy	b
`sym.lu8`	CI statistical info.	t
`mat.HII`	Hamiltonian matrix (nonorth.)	b
`mat.SII`	Overlap matrix (nonorth.)	b
`mat.siok`	Signal for successful S inversion	-
`sm1.mat`	Overlap inverse (nonorth.)	b
`mat.III`	Hamiltonian matrix (orth.)	b
`trn.III`	Symgenn/ortho. basis transf. (orth.)	b
`big.out`	Output file	t
`big.vec`	Eigenvectors	b

The methods of `beigII` are iterative and convergence can be profoundly influenced by the nature of the matrices and the user-specified starting vector. One of two methods is used depending upon the matrix generator. Following `omtrcIII` this module employs the original *Davidson Method*[22]. This is a true N^2 method, but in our experience it is not very efficient when $\mathbf{H}$ is not sparse. After `nomtrcII` and `sinv` the *Gallup Method*[23] is used. It is most efficient for nonsparse matrices or in cases in which $\mathbf{S}$ is far from the identity. It is not, unfortunately, an N^2 method since the inverse of $\mathbf{S}$ is required and the matrix inversion involves an N^3 algorithm. Once $\mathbf{S}^{-1}$ has been found, however, the remainder of the problem is N^2.

As the names of the binary matrix files produced by the orthogonal and nonorthogonal generators are unique, `beigII` chooses its method based upon the presence or absence of files with particular names. With careful cleaning make sure that the only matrix files present are those you intend to diagonalize.

As an initial guess to the ground-state eigenvector, the user must specify basis configurations expected to be significant (but not necessarily the largest) contributors. These must be equal in number to the number of eigenvalues requested and are specified by the position of the symmetry function in the `<prefix>sym.out` list. (Guesses for higher states are derived automatically.) If these principal components are not suggested by the physical nature of the problem, they might be obtained by using `neweig` on a smaller problem including only a subset of the configurations.

Format of the `<prefix>big.inp` File:

Line 1: neig prt_evec

neig	number of eigenvalues to calculate (int, normally ≤ 5)
prt_evec	'0' to skip eigenvector printing '1' to include eigenvectors in the output

Line 2: ibf(1) ibf(2) ... ibf(neig)

ibf(i)	symmetry function expected to contribute to the ground state (int)

5.8 Parameter Optimization

5.8.1 `smplxtrdg`

This module provides a generic and powerful means of optimizing parameters in a calculation with multiple steps. Table 20 contains the names of the files which may be accessed by this module.

Table 20: Files read (above) or written (below) by `smplxtrdg`

Suffix and Extension, us=user-supplied, or (Full File Name), `XXXXXX` is system supplied	Description	Text (t) or Binary (b)
`plx.inp`	User-supplied input	t
us.us	Template module input file	t
us.`out`	Output file with target value	t
us.us	Input file created from the template	t
(`stdout`)	Output of minimization	t
(`sedpXXXXXX`)	Temporary file with sed command	t
(`nrgyXXXXXX`)	Temporary file with awk command	t

Given its relative complexity, this module might have been reserved for "advanced users." It is included, however, mostly because it provides a rudimentary means of optimizing molecular geometry. (The notion that a quantum chemical suite is incomplete without a means of geometry optimization became widespread only after the bulk of CRUNCH was written; `smplxtrdg` is an attempt to bridge that gap.) Illustrations of geometry optimization at several levels of theory are provided among the examples. The general description given here may be difficult to follow without consulting one of them.

Much of the complication arises from the need to change parameters in the input file to one module, gauge the effect on a target value which can be in the output of another module, and then repeat the calculation with new input parameters chosen to minimize the target (often an energy). Not only are multiple CRUNCH modules linked together in this way but, in the interests of flexibility, the input parameters to non-CRUNCH programs called before CRUNCH can be similarly optimized. Thus Table 20 may appear deceptively simple. Although not directly manipulated by `smplxtrdg`, the user must supply all of the other input files required by modules in the chain including input and target.

There is considerable freedom in naming the files used by `smplxtrdg` but good practice would suggest that a suitable suffix and extension for the template to a `<prefix>lob.inp` file might be `<prefix>lob.tmp`. The template file is a complete input file but with the parameters to be varied replaced by an alphanumeric label unique in the file. The `sed` command is used to substitute the input parameter for the label. After the chain of modules has been executed, the `awk` command is used to extract the target value. (Intermediate files with names made unique by incorporation of a system-generated pseudorandom string, represented by `XXXXXX` in Table 20, are created in carrying out these commands.)

The algorithm used to suggest new input parameters and verify the minimal solution is the multidimensional simplex method (see [21], section 10.4 for a short discussion). Although our account and the examples emphasize geometry optimization it is worth mentioning that `smplxtrdg` is sufficiently general that it can optimize any parameters given in a single input file.

This module is not incorporated into the crunch script but must be called directly in the manner of:

```
    smplxtrdg h2.plx.inp > h2.plx.out
```

where the standard output has been redirected to an external file where it is available for later reference.

Format of the `<prefix>plx.inp` File:

The input data are divided into the sections given below. Line numbers are internal to these sections.

- Commands before CRUNCH

  Line 1: nbc

nbc	number of commands to execute before CRUNCH modules (int)

  Next nbc lines: npcomm(i)

npcomm	command number i to execute before CRUNCH (string, < 127 char.)

- Input and template files

  Line 1: inpend

inpend	suffix and extension of input file to receive inputs (string, < 127 char.)

  Line 2: tmpend

tmpend	suffix and extension for template of input file (string, < 127 char.)

- CRUNCH command

  Line 1: crcom

crcom	command to execute chain of CRUNCH modules (string, < 127 char.)

- Output manipulation

  Line 1: awkc

awkc	awk command to extract target value from output (string, < 127 char.) This should be surrounded by single quotation marks.

  Line 2: outend

outend	suffix and extension of output file with target (string, < 127 char.)

- Parameters to Optimize

  Line 1: npar

npar	number of input parameters to optimize (int)

  Next npar lines: parlab(i) initval(i) delta(i)

parlab(i)	label standing in for parameter i in template (string, $\leq$ 20 char.)
initval(i)	initial value for parameter i (float)
delta(i)	initial increment for the parameter (float)

6 Example Problems

There are a number of example problems supplied with the source distribution (see the subdirectories beneath `examples`). When a systemwide installation of CRUNCH has been performed, this directory can be found just below the directory containing this manual. In addition to the monograph[3] these example problems should be of particular help to a new user. Each module (but perhaps not each option of each module) is exercised by at least one of the examples. To help confirm proper module operation, reference output files (with names ending in `.sve`) are included, in addition to the input files used to generate them. Beginners should work through the examples or, at least, find the example closest to their problem of interest and use it as illustration and template.

Current example systems include the boron atom, the H_2, BF, BO, HF, F_2, NH, and O_2 diatoms, and the polyatomic molecules/ions, CH_2, CH_4, naphthalene, and imidazolium cation. See `README.examples` in the `examples` directory for a matrix detailing the modules used by each example. Each problem also comes with a separate `README` file (*e.g.* `examples/hf/dunning/README.hf`) which gives further details about the calculation, including which commands are needed to generate the output files.

7 Copyright and License

8 Limitations and Bugs

Perhaps the most serious scientific limitation is the lack of provision for lobe-Gaussian orbitals with $l > 3$. Of course, off-center Gaussians have been used to simulate large angular momentum functions.

Currently, averaging across iterations and energy-shifting the virtual-orbital eigenvalues of the Fock operators are the only means of improving convergence in the main single-configuration R(O)HF module (`gscfnn`). As a result, SCF convergence can be somewhat slow and occasionally even unattainable. For systems with appropriate spin the `mxsscf` module allows somewhat greater control over convergence.

CRUNCH also lacks automatic, tailored geometry optimization. As a partial, and not especially efficient, solution to this, the `smplxtrdg` module, agnostic to the nature of the parameters, is provided and can be used to find the optimal value of any number present in an input file. Expert users can even use this capability to perform certain types of quantum-chemical calculations not explicitly supported by CRUNCH.

Almost certainly, there are bugs. Nevertheless, the various modules have, over the years, been subjected to vastly different amounts of usage. In our opinion, of the primary modules, the integral and single-configuration R(O)HF modules are the best tested, followed by `trannn`, `symgenn`, the original matrix generators, and the diagonalizers. The linear-molecule capabilities are some of the newest, the `mp2` and `mcscf` modules and the new matrix generators have received a modest amount of successful usage, and `smplxtrdg`, because of its generality, can be somewhat temperamental.

Bug reports and questions, in general, should be directed to us at the e-mail addresses on the title page.

9 Acknowledgments

These programs have been worked on by a number of individuals in addition to the authors. We would like to acknowledge Joe Norbeck, Bob Vance, and Jack Collins, who have made significant original contributions. JDM would also like to acknowledge the Air Force Office of Scientific Research under whose Visiting Scientist Program some of the more recent work was performed. During this time Peter W. Langhoff, Michal Ben Nun, Michael Bromley, and Kyle Rollin of the Univ. Calif.-San Diego and Spectral Associates, LLC provided invaluable interaction and feedback.

10 References

[1] J.N. Murrell, S.F.A. Kettle, and J. M. Tedder, *The Chemical Bond*, (Wiley, New York, 1985).

[2] W.J. Hehre, L. Radom, P.v.R. Schleyer, and J.A. Pople, *Ab Initio Molecular Orbital Theory*, (Wiley, New York, 1986).

[3] G.A. Gallup, *Valence Bond Methods*, (Cambridge Univ. Press, Cambridge, 2002).

[4] F.A. Matsen, Adv. Quantum Chem., **1**, 60 (1964).

[5] T. Clark, *A Handbook of Computational Chemistry*, (Wiley, New York, 1985).

[6] M.S. Gordon and M.W. Schmidt, *Theory and Applications of Computational Chemistry*, edited by C.E. Dykstra, G. Frenking, K.S. Kim, and G.E. Scuseria, (Elsevier, Amsterdam, 2005).

[7] G.A. Gallup, Int. J. Quantum Chem. **8**, 267 (1974), esp. Sect. 2.

[8] C. Trindle and D. Shillady, *Electronic Structure Modeling: Connections Between Theory and Software*, (CRC, Boca Raton FL, 2008), pp. 61-76.

[9] E.R. Davidson and D. Feller, Chem. Rev. **86**, 681 (1986).

[10] R.E. Watson, Phys. Rev. **111**, 1108 (1958).

[11] G.A. Gallup, J. Chem. Phys. **45**, 2304 (1966)

[12] J.F. Rico, R. López, A. Aguado, I. Ema, and G. Ramiŕez, Int. J. Quantum Chem. **81**, 148 (2001). See also, J.R.Á. Collado, J.F. Rico, R. López, M. Paniagua, and G. Ramacuteirez, Comp. Phys. Comm. **52**, 323 (1989).

[13] C.C.J. Roothaan, Rev. Mod. Phys. **23**, 69 (1952); C.C.J. Roothaan, Rev. Mod. Phys. **32**, 179 (1960).

[14] R. Carbo and J.M. Riera, *A General SCF Theory*, (Springer-Verlag, Berlin, 1978).

[15] M.F. Guest and V.R. Saunders, Mol. Phys. **28**, 819 (1974).

[16] A. Szabo and N.S. Ostlund, *Modern Quantum Chemistry*, First Rev. Ed., (McGraw-Hill, New York, 1982).

[17] G.A. Gallup, R.L. Vance, J.R. Collins, and J.M. Norbeck, Adv. Quantum Chem. **16**, 229 (1982).

[18] C. Murray and E.R. Davidson, Chem. Phys. Lett., **187**, 451 (1991).

[19] K.R. Glaesemann and M.W. Schmidt, J. Phys. Chem. A, **114**, 8772 (2010).

[20] H.-Y. Werner and W. Meyer, J. Chem. Phys. **73**, 2342 (1980); H.-Y. Werner and W. Meyer, J. Chem. Phys. **74**, 5794 (1981).

[21] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Second Ed., (Cambridge Univ. Press, Cambridge, 1992).

[22] E.R. Davidson, J. Comp. Phys., **17**, 87 (1975).

[23] G.A. Gallup, J. Comp. Chem., **3**, 127 (1982).

11 Appendix with Common SCF Parameters

We give here a few common examples of the spin-coupling parameters, W, α, and β. The molecular orbitals are grouped into shells according to the Fock operator which describes them and the coupling parameters circumscribe the interaction of these shells. The parameters are vectors or matrices, $\mathbf{W}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$, indexed by shell. Only common cases with no more than three independent Fock operators are given here. (See reference [14] for a more exhaustive list.)

- Closed-shell singlets:

  These values have already been given, but are repeated for reference. There is only one Fock operator and so $\mathbf{W}$ has one component and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are 1×1 matrices.

$$\begin{aligned}\mathbf{W} &= [2.0]\\ \boldsymbol{\alpha} &= [2.0]\\ \boldsymbol{\beta} &= [1.0]\end{aligned}$$

- Open-shell, high-spin systems (including a doublet with one singly occupied orbital):

  When each of one or more nondegenerate orbitals are occupied with a single electron and the spins of the electrons are aligned, there are two Fock operators, the first for the underlying closed-shell and the second for the open-shell.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 2.0 & 1.0 \end{bmatrix}\\ \boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 0.5 \end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}\end{aligned}$$

If some of the open-shell orbitals are degenerate, these parameters must be used with special care in order to avoid symmetry-broken solutions. In such cases the *state-averaged* parameters (see an example below) must be used to enforce symmetry.

- A first-excited singlet:

  This is the type of state which would result from spin-preserving promotion of one electron of a closed-shell singlet into an unoccupied orbital. There are three independent Fock operators, the first for the doubly occupied orbitals remaining and the remaining two for each of the open shells.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 2.0 & 1.0 & 1.0 \end{bmatrix}\\ \boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.0 & 1.0 \\ 1.0 & 0.0 & 0.5 \\ 1.0 & 0.5 & 0.0 \end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.5 & 0.5 \\ 0.5 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.0 \end{bmatrix}\end{aligned}$$

- An open, doubly degenerate shell (above a closed-shell base):

  Whether the two orbitals of the open shell are occupied by one, two, or three electrons, there are two Fock operators, the first for the closed-shell and the second for the open-shell. W_2 in each case is 2f where f is the filling fraction.

  One electron yields a ^{2}E state.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 2.0 & 0.5 \end{bmatrix}\\ \boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 0.5 \\ 0.5 & 0.0 \end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.25 \\ 0.25 & 0.0 \end{bmatrix}\end{aligned}$$

  Two electrons can give three states: ^{3}A, ^{1}A, and ^{1}E (see, for example, [11]).

  All states have:

$$\mathbf{W} = \begin{bmatrix} 2.0 & 1.0 \end{bmatrix}$$

  For ^{3}A:

$$\begin{aligned}\boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 0.5 \end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}\end{aligned}$$

  For ^{1}A:

$$\begin{aligned}\boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 0.0 \end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}\end{aligned}$$

For ^{1}E:

$$\begin{aligned}\boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 0.25 \end{bmatrix} \\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 0.0 \end{bmatrix}\end{aligned}$$

Three electrons yields a ^{2}E state which is the "hole analog" of the one-electron case.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 2.0 & 1.5 \end{bmatrix} \\ \boldsymbol{\alpha} &= \begin{bmatrix} 2.0 & 1.5 \\ 1.5 & 1.0 \end{bmatrix} \\ \boldsymbol{\beta} &= \begin{bmatrix} 1.0 & 0.75 \\ 0.75 & 0.5 \end{bmatrix}\end{aligned}$$

- A state-averaged (sp) triplet:

  To obtain, for example, the isotropic, excited ^{3}P state of beryllium (nominal configuration, $1s^2 2s2p$), state-averaging is necessary. With a doubly occupied base, there are three Fock operators and the required parameters are:

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 2 & 1 & 1/3 \end{bmatrix} \\ \boldsymbol{\alpha} &= \begin{bmatrix} 2 & 1 & 1/3 \\ 1 & 0 & 1/6 \\ 1/3 & 1/6 & 0 \end{bmatrix} \\ \boldsymbol{\beta} &= \begin{bmatrix} 1 & 1/2 & 1/6 \\ 1/2 & 0 & 1/12 \\ 1/6 & 1/12 & 0 \end{bmatrix}\end{aligned}$$

Other state-averaged calculations may require even more Fock operators. In their Appendix A Carbo and Riera[14] give another simple example and the author of [13] gives a number which are not as simple (but readers are cautioned that the spin-coupling parameters in [13] are defined somewhat differently from the Carbo-Riera parameters used in CRUNCH).